1980 APL Users Meeting

October 6, 7 and 8
Hotel Toronto
Toronto, Canada

Sponsored by:

I. P. Sharp Associates Limited

APL Users Meeting

Toronto

October 6, 7, 8, 1980

Sponsored by:
I. P. Sharp Associates Limited

CONFERENCE PROCEEDINGS

LIST OF PAPERS BY SESSION

DAY 1

MORNING

General Session

AFTERNOON
Session A: **Managing APL Resources**

Session B: **APL For Financial Applications**

Session C: **Applications in the Aviation Industry**

DAY 2

MORNING
Session A: **International Networks For Corporate Communications**

Session B: **APL For Planning Forecasting and Economics**

Session C: **Personnel and Record Handling Systems**

AFTERNOON

Session A: **In-House APL Time-Sharing Services**

Session B: **APL For Decision Support Systems**

Session C: **Private Data Bases**

Session D: **A Derivative Algorithm for APL**

DAY 3

MORNING

Session A: **APL Techniques and Programming Tools**

Session B: **Specialized APL Applications**

Session C: **Applications in the Energy Industry**

AFTERNOON

General Session — Teaching APL

EVENING TECHNICAL SEMINARS

(PART 1)
Session A: **Data Manipulation and Display Techniques**

Session B: **Arrays**

Session C: **Electronic Mail**

Session D: **Event Handling**

(PART 2)
Session A: **Forecasting and Econometric Modelling**

Session B: **Shared Variables**

Session C: **Record Handling Systems**

Session D: **The Package Data-Type**

*Indicates papers were not available at time of printing

LIST OF PAPERS BY AUTHOR

COST MANAGEMENT IN A GROWING APL DEPARTMENT

Barry M. Linton
Rank Xerox Ltd.
London, England

Since the early 1970s, the growth in Rank Xerox usage of APL has been explosive and it still continues to grow steadily into the 1980s. We now have 350-400 personnel whose familiarity with APL ranges from the minimum required to run existing systems to the high level required to develop these systems. This period of growth has also seen the devolution of much computer usage from the confines of the traditional data processing department directly into the hands of the end users. Such has been my own path within Rank Xerox. At one time I was responsible for the development of systems within our Corporate Information Systems Department. Now I am controlling a group who use APL extensively to analyse and forecast the Corporation's marketing performance and to develop future plans.

The problems of managing this type of activity are somewhat different to those I had faced earlier. In the traditional "Cobol" shop, the main management concerns were to:

1. Assess the labour content required for the various phases of systems design, programming, testing and full documentation.

2. Balance the available staff to the resources required, giving due consideration to the total project timescales and other work required from the department.

3. Continually monitor and control the stages of completion against the plan.

In addition to one's own experience, there were many "rules of thumb" and algorithms based on processing complexity and the number of files, etc., that could be used to forecast the resources for programming and program testing. Computer usage and data preparation costs could also be estimated in a fairly accurate manner.

Like so many companies' Data Processing Departments, we worked to a zero-based budget in which all the labour and computer costs were re-charged back to sponsoring end-user departments. Although Financial Controllers and accountants would disagree, the end-users' normal attitude to such re-charges are that they consist of "funny-money", in as much as costs were transferred from one department to another.

In our APL group, as with other similar groups in Rank Xerox and within other companies, the management problems and controls are different. To start with, only one of our six people who use APL is a full-time APL programmer. Each individual is using APL because of its productivity as a tool to solving specific business problems. Therefore, the issue is now far less one of labour cost but one of controlling the costs of our APL timesharing bills. Also, as these bills are being paid directly to an outside company, they are anything but "funny-money" and any saving will show directly on our operating profit. I am aware of this, and so too are our senior management who have to approve the annual budgets!

As I have already stated, Rank Xerox's growth in APL usage has been explosive and this has manifested itself not only by departments submitting higher and higher budgets for approval each year, but also by being regularly well overspent against that amount which had been previously allocated.

Therefore, at the last APL user's meeting here in September 1978, with my own budget 50% overspent, I asked the following questions of those assembled.

"How many people are overspent on their budget?"

"How do they plan their budget for the next year?"

"How do they control their expenditure during the year?"

The initial response was a widespread embarrassed sympathy (including our conference hosts!). There were no conclusive answers, although there were two comments that I do recollect; one delegate saying that he always had the calculated cost of each report page printed at the bottom, and another who considered the number of terminals he had as being a limiting factor on his expenditure. Unfortunately, in my own case, each of these suggestions would not completely help since a significant proportion of our costs are taken up by data storage and by B/N-tasks which do not pass through the terminal.

However, help was soon at hand when, in November 1978, Rank Xerox's Central Timesharing Support Group developed and made available a system for the Corporation's world-wide APL users. It was called APL Cost Management System (known as ACMS). The implementation and regular use of this system, together with the application of a few simple financial control techniques, has enabled me, over the past two years, to take a far greater hold over our APL expenditure levels.

ACMS is basically a reporting system that accesses the IPSA USAGE files to calculate current billing and also holds a historical file of the billing of all Rank Xerox account numbers. (At the present time Rank Xerox has about 1,100 account numbers allocated, of which about 800 are regularly active.)

The reports can be produced at a variety of hierarchies within the organisation, as can be seen in Figure 1.

In general, the higher level is able to see a roll-up report of the costs being incurred at the level(s) below, including project and cost centre details. The System Steward (SS), who is based at the International HQ, is responsible for setting up and maintaining the ACMS parameters (e.g., IPSA charging rates, currency conversion rates, permission access for the DGS, DSS, PC etc.).

ACMS HEIRACHY

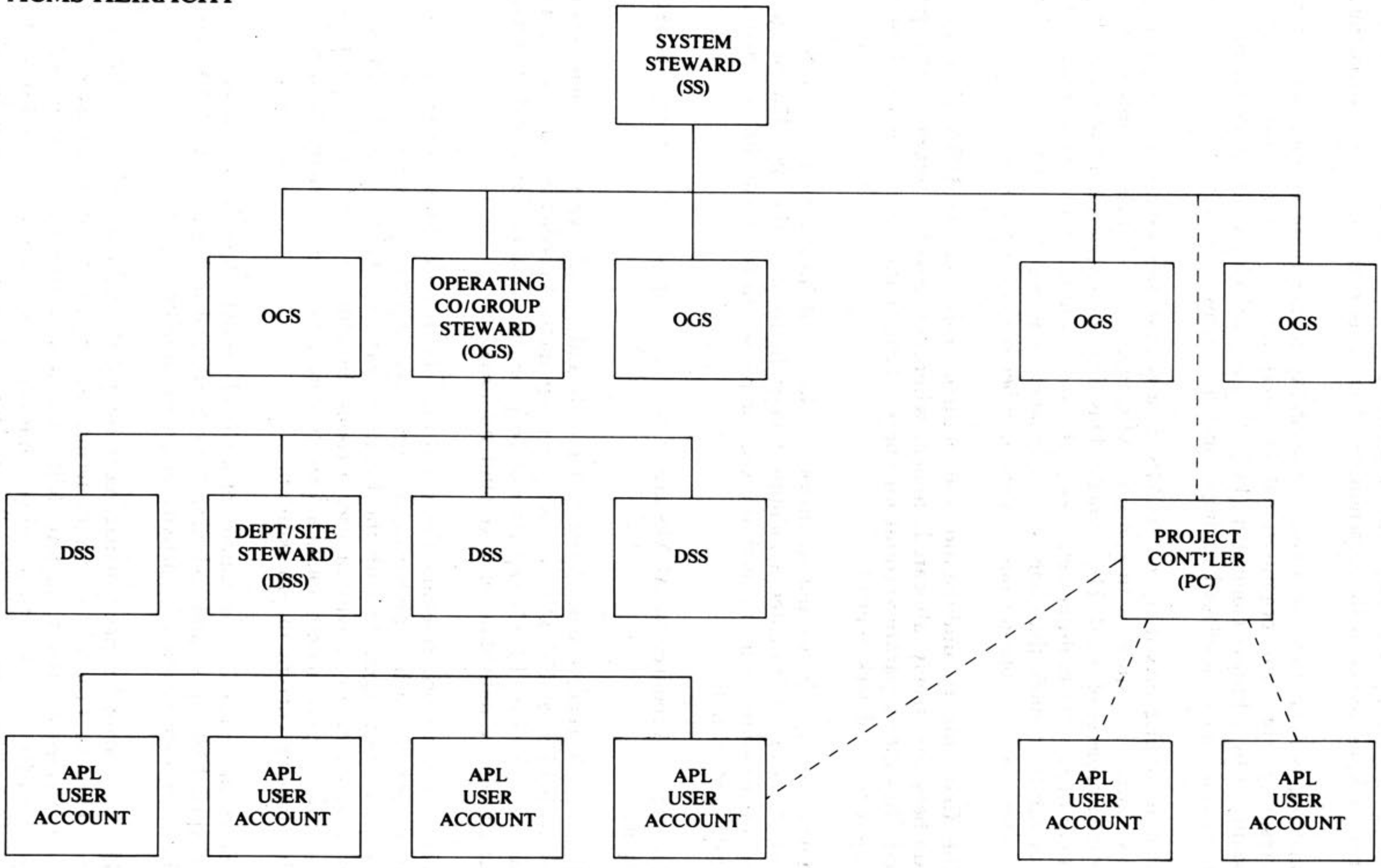


Figure 1

The Operating Company/Group Steward is able to monitor all costs within the departments of his Operating company. The costs can be expressed in local currency, dollars or sterling. The Department Steward (DSS) level is the level that I operate, where I have access to the expenditure of each user account in the department.

Superimposed on this hierarchy is the ability to nominate various account numbers (irrespective of their organisational location) into a named project which can be monitored by a Project Controller (PC). This is particularly useful for the management of internationally developed or implemented systems.

Prior to setting myself up on ACMS, it was first necessary to try to define all our major activities, since, with end-user APL-type work, it may be less clear than the defined projects of a DP Department. This definition of activities included those cases where there were multiple activities in the account number and those where an activity was run from more than one account number. The objective, therefore, was, wherever possible, to get a one-for-one account number/activity set up.

The first rule to understand and control costs: on the IPSA system, account numbers are freely allocated, hence, wherever possible, segment the projects/activities into separate account numbers. (Temper the proliferation with due regard to security and lock control.)

Having set up the account numbers to relate to defined activity, ACMS also allows each DSS or PC to enter a monthly budget figure onto the file. The budget can be an equal monthly sum through the year or phased to represent the expected workload peaks and troughs.

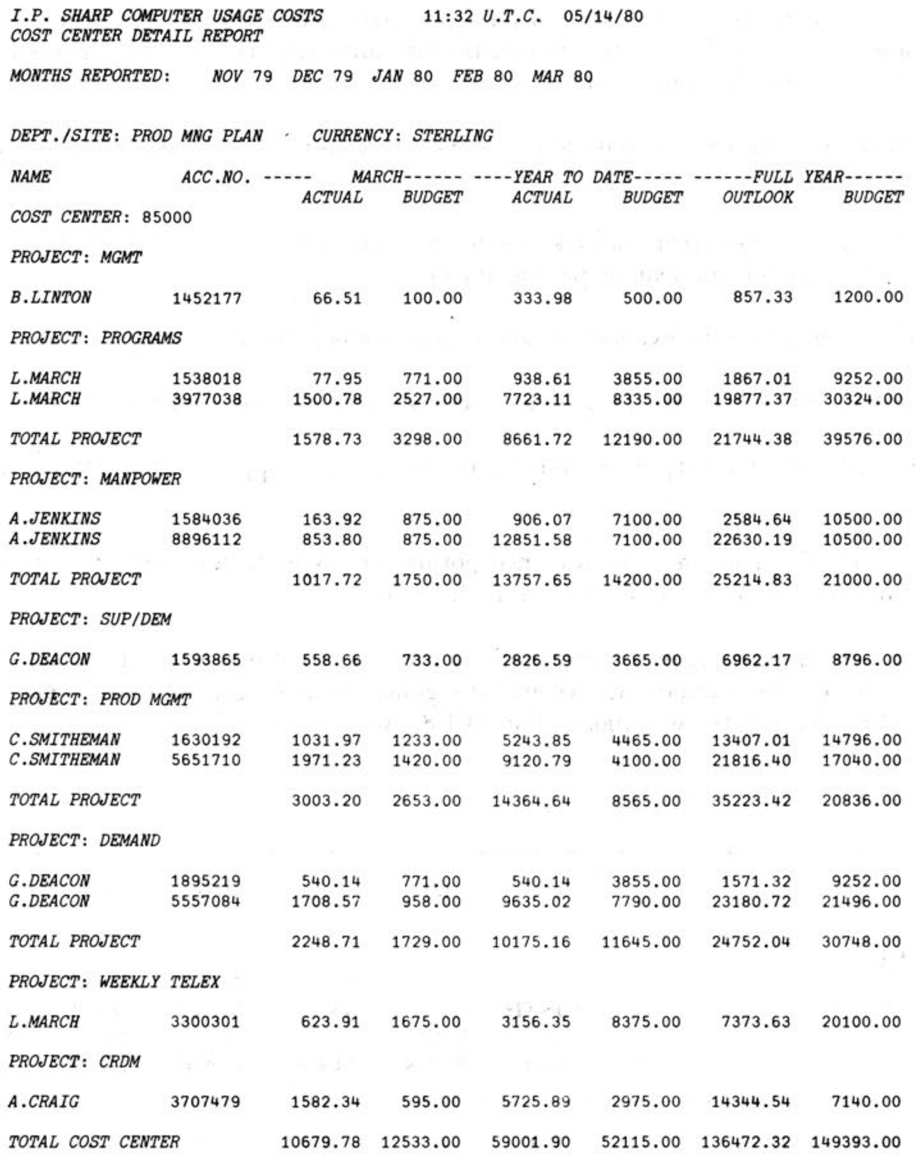
The reports produced by ACMS are of two main types — PLAN reports and COST reports.

The PLAN reports show latest full month and cumulative year to date expenditure versus budget, and also, by a moving average method predict an outlooked expenditure for the full year. PLAN reports can be produced as a departmental summary or by activity or by individual user account numbers.

The COST reports resemble the actual invoices that I.P. Sharp issue, in that the costs attributed to each timesharing resource are separately shown. The total cost is compared to budget. As with the PLAN reports, the COST reports can be produced as a summary, by activity or user account. In addition, the COST report can be run to show a given month total, a year to date total and, particularly useful, a report showing the current month position.

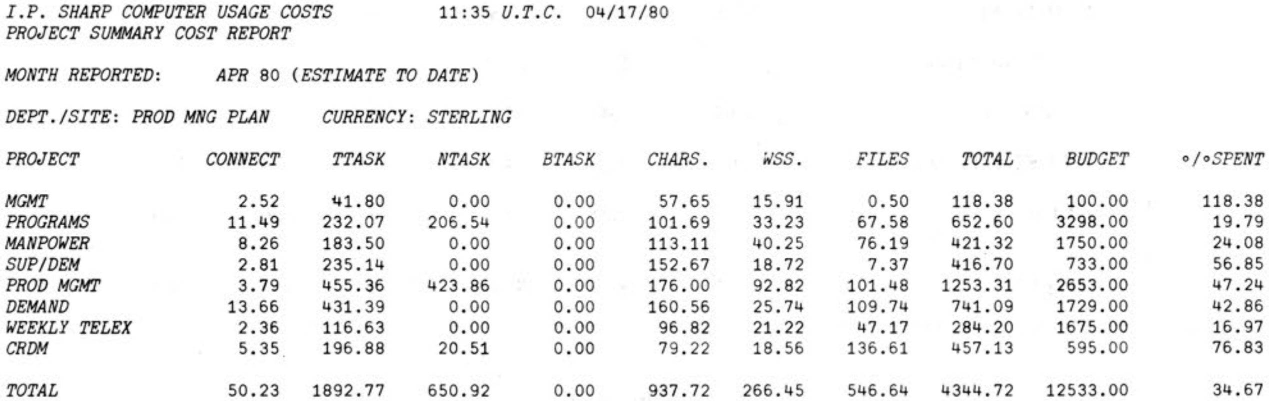
Shown in Figure 2 are examples of a COST report which shows, by activity, the costs in April 1980 up to and including 16th April, and in Figure 3 a PLAN report giving the budgetary position by activity and user numbers.

The method of financial management control throughout Xerox and Rank Xerox is the "Outlook Process". In simple terms, this means that every management group or Cost Centre in the Company will each month prepare a considered (and often committed!) outlook of their full year business situation. For a sales district, the outlook would be net orders and installations; for a staff group it would be costs (salaries, overheads, travel, data processing costs, etc.). Each Operating Company will roll this up to get revenue and the outlooked profit.



```
I.P. SHARP COMPUTER USAGE COSTS          11:32 U.T.C.  05/14/80
COST CENTER DETAIL REPORT

MONTHS REPORTED:     NOV 79  DEC 79  JAN 80  FEB 80  MAR 80


DEPT./SITE: PROD MNG PLAN     CURRENCY: STERLING

NAME             ACC.NO. -----    MARCH------ ----YEAR TO DATE----- ------FULL YEAR------
                              ACTUAL    BUDGET     ACTUAL     BUDGET    OUTLOOK     BUDGET
COST CENTER: 85000

PROJECT: MGMT

B.LINTON         1452177       66.51    100.00     333.98     500.00     857.33    1200.00

PROJECT: PROGRAMS

L.MARCH          1538018       77.95    771.00     938.61    3855.00    1867.01    9252.00
L.MARCH          3977038     1500.78   2527.00    7723.11    8335.00   19877.37   30324.00

TOTAL PROJECT                1578.73   3298.00    8661.72   12190.00   21744.38   39576.00

PROJECT: MANPOWER

A.JENKINS        1584036      163.92    875.00     906.07    7100.00    2584.64   10500.00
A.JENKINS        8896112      853.80    875.00   12851.58    7100.00   22630.19   10500.00

TOTAL PROJECT                1017.72   1750.00   13757.65   14200.00   25214.83   21000.00

PROJECT: SUP/DEM

G.DEACON         1593865      558.66    733.00    2826.59    3665.00    6962.17    8796.00

PROJECT: PROD MGMT

C.SMITHEMAN      1630192     1031.97   1233.00    5243.85    4465.00   13407.01   14796.00
C.SMITHEMAN      5651710     1971.23   1420.00    9120.79    4100.00   21816.40   17040.00

TOTAL PROJECT                3003.20   2653.00   14364.64    8565.00   35223.42   20836.00

PROJECT: DEMAND

G.DEACON         1895219      540.14    771.00     540.14    3855.00    1571.32    9252.00
G.DEACON         5557084     1708.57    958.00    9635.02    7790.00   23180.72   21496.00

TOTAL PROJECT                2248.71   1729.00   10175.16   11645.00   24752.04   30748.00

PROJECT: WEEKLY TELEX

L.MARCH          3300301      623.91   1675.00    3156.35    8375.00    7373.63   20100.00

PROJECT: CRDM

A.CRAIG          3707479     1582.34    595.00    5725.89    2975.00   14344.54    7140.00

TOTAL COST CENTER           10679.78  12533.00   59001.90   52115.00  136472.32  149393.00
```

Figure 2



```
I.P. SHARP COMPUTER USAGE COSTS          11:35 U.T.C.  04/17/80
PROJECT SUMMARY COST REPORT

MONTH REPORTED:      APR 80 (ESTIMATE TO DATE)

DEPT./SITE: PROD MNG PLAN      CURRENCY: STERLING

PROJECT            CONNECT     TTASK     NTASK     BTASK     CHARS.     WSS.     FILES     TOTAL     BUDGET      ∘/∘SPENT

MGMT                  2.52     41.80      0.00      0.00      57.65    15.91      0.50    118.38     100.00        118.38
PROGRAMS             11.49    232.07    206.54      0.00     101.69    33.23     67.58    652.60    3298.00         19.79
MANPOWER              8.26    183.50      0.00      0.00     113.11    40.25     76.19    421.32    1750.00         24.08
SUP/DEM               2.81    235.14      0.00      0.00     152.67    18.72      7.37    416.70     733.00         56.85
PROD MGMT             3.79    455.36    423.86      0.00     176.00    92.82    101.48   1253.31    2653.00         47.24
DEMAND               13.66    431.39      0.00      0.00     160.56    25.74    109.74    741.09    1729.00         42.86
WEEKLY TELEX          2.36    116.63      0.00      0.00      96.82    21.22     47.17    284.20    1675.00         16.97
CRDM                  5.35    196.88     20.51      0.00      79.22    18.56    136.61    457.13     595.00         76.83

TOTAL                50.23   1892.77    650.92      0.00     937.72   266.45    546.64   4344.72   12533.00         34.67
```

Figure 3

Within my own group, the external data processing costs are as significant as salaries and other costs, and I exercise controls by the same approach, which means following these basic principles each month.

1. Understand the costs to date and be able to explain all the variances from plan for each activity.

2. Develop a rest-of-year outlook based on what has happened, plus any actions to be taken in the immediate period ahead.

3. Add to year-to-date expense to get a new outlook for the year.

4. Explain the variations, by activity, between this outlook and the prior outlook.

5. Quantify all unincorporated but perceived risks or opportunities around the final outlook.

As a general principle, always incorporate or explain any changes in risks or opportunities between one outlook and the next.

Figure 4 is an example of a recent outlook exercise. In Rank Xerox parlance, the term "5+7" means the outlook at five months gone, seven to go; our fiscal year starts in November. Therefore, we update the "4+8" with a "5+7".

	(1)	(2)	(3)	(4)	(5)	(6)
£k				5 + 7		
			REST	FULL	B/(W)	B/(W)
	MONTH		YEAR	YEAR	4 + 8	FULL
	5	B/(W)	OUT-	OUT-	OUT-	YEAR
	YTD	PLAN	LOOK	LOOK	LOOK	PLAN
MANAGEMENT	0.3	0.2	0.4	0.7	—	0.5
PROGRAMS	8.7	3.5	10.0	18.7	5.0	20.8
MANPOWER	13.8	0.4	27.0	40.8	—	(19.8)
SUP/DEM	2.6	1.0	3.9	6.5	—	2.3
PROD MGMNT	14.3	(5.7)	10.0	24.3	4.0	(3.6)
DEMAND	10.2	1.4	14.1	24.3	—	6.7
WEEKLY TELEX	3.1	5.2	4.4	7.5	1.4	12.6
CRDM	5.7	(2.8)	16.0	21.7	(6.0)	(14.6)
TOTAL	58.7	3.2	85.8	144.5	6.4	4.9

Figure 4

Column (6) is the variance analysis to plan. It clearly reflects the decision earlier in the year to embark on a new project on manpower analysis. The savings in "Programs" were due to implementing a more economical file structure.

Outlook changes in the 5+7 were due to:

1) Cancellation of part of Program activity.

2) Management agreement to run Product Management suite at lower frequency due to cost over-run.

3) CRDM support carried out in more depth than prior year on which plan was based. (Incorporating risk in 4+8 outlook).

4) Savings in weekly Telex activity, due to decision to drop further part of analyses.

NOTE:

> Unincorporated opportunity of 2.0K in Product Management due to exercise to improve program efficiency.

Figure 4 would represent the approach made to present the budget situation to my management. It is important to notice that economy actions such as file deletions, workspace control, use of B/N-tasks etc., are considered to be a routine discipline by all APL users to keep costs down, although there is a technical factor that I like to look at, which is the cost ratio of CPU units to character charge. For the type of work we do, I expect to see a ratio of 2.5 - 4 to 1 based on the I.P. Sharp U.K. billing rates. In cases where this ratio is significantly higher, I will try to find out if this has been due to inefficient programming or testing activity. Naturally the ratios will differ for other types of programming, e.g., interactive modelling/simulation, but the principle is the same - if the ratio varies over a period, find out why. However, I must emphasise that the outlook process is to clearly see the impacts of decisions previously made and of any actions planned.

Having kept the timesharing bills under control, the manager must also apply these same techniques to terminal rentals, communication costs, ancillary supplies and last, but definitely not least, contract programmers.

Success in keeping to budget also depends very much on the ability to plan the required budget at the appropriate time.

A basis for developing any plan for the following year is the full understanding of the present expenditure. If one was to start using the process I have described from the start of a financial year, it should be quite possible to develop a reasonable and supported budget plan for the following year, as the plan development will usually occur around the time when you have eight to nine months of actual data available.

The difficulty will be assessing the costs of new activities for the next year but you will be helped here by knowing the relative costs of similar projects and the productivity of the individual APL users. It is worth mentioning that the Level of Service now experienced on the I.P. Sharp Service in the U.K. is sufficiently high to ignore the cost effects of downtime, re-runs or staff overtime. This may not necessarily be true in the more remote regions of the network or on other vendor's services.

And, as a final word of warning, — **don't overcall your budget** — it can lead to a slackening of cost control discipline and embarrassing questions being asked.

SHARP APL SYRACUSE

Dana E. Cartwright
Phyllis A. Kent
Academic Computing Center
Syracuse University
Syracuse, New York

Abstract

For more than a decade Syracuse University has provided APL to its academic community. However, new APL features developed elsewhere in those years were not made available, and Syracuse was left with an out-of-date APL system. The decision to bring SHARP APL to Syracuse was prompted by an urgent need to make a modern APL available on campus. This paper describes the environment in which the conversion to SHARP APL took place. It emphasizes the impact on users and Computing Center staff as the installation progressed. The special needs of academic people, the desire to retain local APL enhancements, limited machine access, a non-standard operating system, and a community of 7000 APL users all conspired to make this a challenging project.

Introduction

Any discussion of the installation of SHARP APL at Syracuse University must be prefaced by a brief history of our involvement with APL and a review of the special needs and expectations of our faculty and students. While it would be relatively easy to install APL in a sterile environment which had no users, to bring a new APL system to Syracuse, where APL is as vital as food and water, was intimidating at best.

This historical perspective begins in late 1967 when about twenty faculty and graduate students explored APL using a single terminal with phone access to IBM at Yorktown Heights, and continues with growing interest in APL and increased system availability. A DOS version of APL was limited to four hours a day in 1968. In 1969 APL became available full time from thirty-four terminals under the Syracuse University Operating System (SUOS), a modified OS/MFT. We were fortunate to obtain an early version of APL*PLUS from Scientific Time Sharing Corporation in the early 1970's. After the File Subsystem was installed a short time later, the APL boom was on at Syracuse.

APL, A Homegrown Product

From the beginning, responsibility for installing and maintaining APL belonged to the staff of the Academic Computing Center. It would be safe to say that Syracuse University never ran an "off the shelf" APL system with regular releases from any vendor and never depended on outside support. In true university tradition, the systems team responsible for our APL added a number of homegrown enhancements, and the users grabbed each new feature as it came along.

We had spent ten years growing our own excellent system, and by 1978 it was clear that the move to any other modern APL would be a major task. Not only were our enhancements burdensome to change, but the sheer size of our user community was awesome. At the time of our conversion, we were faced with 7300 active user accounts running 30,000 terminal tasks a month. These accounts owned 5400 workspaces and 4000 files. At least 20 percent of these users were fully dependent on features of our APL which were not compatible with any other APL system.

The University: An Unreal World

A comparison between the computing needs of a commercial organization and those of a large university shows the nature of our peculiar problem. In business and industry, computing is usually confined to a set of well-defined tasks which support the efficient management of the organization. Someone in the organization is aware of and accountable for the computing being done. While there is accounting for academic computing in departments and colleges at Syracuse, it is not appropriate or feasible for an academic computing center to review or judge the great variety of systems and programs developed by faculty, students and administrators.

The environment in which our users have lived with APL for twelve years has fostered many ingenious APL creations. These range from systems of great complexity to simple but elegant solutions to very special kinds of technical problems. A few APL packages were developed by people who have long since departed the scene. Some programs may be vital cogs in a long-term research project or in the regular administrative function of the University.

Academic Computing — Some Examples

As staff of the Academic Computing Center, we see very few details of the packages written by our users. We give advice and diagnostic help when asked, and in that way, discover some of the interesting problems being solved. Occasional surveys bring to light other programs and systems, and as a preface to the conversion, we conducted a survey to discover new applications. Some examples of current APL projects illustrate the variety of those that existed at the time we planned for the conversion.

Among those used by faculty are course tutorials, computer generated exams, student assignments, and course records kept by individual instructors. Students must do regular course assignments using APL and complete term projects which may also require text edited papers. Students, with unending ingenuity, have involved APL in their social lives. With APL accounts provided by the Office of Student Affairs, programs have been developed to dispense fraternity phone bills, provide a system of ride service for students wishing to share transportation, and to distribute an apartment and room rental guide for the university area.

While most University administrative computing is done on a separate computer, APL on the Academic computer provides an important conversational interface to a number of on-going university computing functions. Examples are APL functions that do the billings to students for their meal plans and housing, and the daily data entry for the library information systems. Deans and Department Chairmen keep a number of records in APL files, such as office inventories, faculty schedules, and budget information. Faculty at remote sites require the APL Internal Job Reader, one of the most heavily used local enhancements, to submit batch jobs.

All of these applications depend on a reliable APL system at some crucial time in the academic term. As we planned for the conversion, it became apparent that there was no time that was not critical to someone. We hoped that the brief period between semesters would be least disruptive, but even then we found that preparation for the new term required APL to be alive and well.

The Decision

It was not easy to decide which APL to install. There were certain to be major problems with any system we chose. We were completely dependent on two features of APL*PLUS. These were the file subsystem and the formatting primitive *ΔFMT*. IBM's APLSV was rejected because it did not have these. Any system which would provide them was a prime candidate for us. Other considerations in our choice of a product were: dedication of the company to continuing support of APL, experience with in-house installations, and a regular program of system enhancements. DEC's APLSF, which runs on our KL-10, was eliminated as a full substitute for our major APL system, because DEC APL support was not up to our standards and the file system was not completely compatible.

In the end, the choice was clearly between STSC and I.P. Sharp, and it was not an easy choice. We were finally persuaded to install SHARP APL by the offer of substantial installation support, continuing support for an OS/MVT based APL, and regular releases of the system containing the latest APL enhancements.

Going It Alone

The adaptation of the SHARP APL system to run at Syracuse University was done entirely using our own staff. In addition, much of the preparation and adaptation of Sharp's conversion software to SU's special needs was done in-house. Sharp offered considerable additional manpower to assist in these efforts, but SU declined the offer. Several considerations motivated the decision to take this approach.

Firstly, because the operating system at SU is non-standard, the Sharp software would have to be modified to make it run at SU. In general, the modifications might involve changes to APL code or changes to the operating system. In such a situation, it seemed desirable to us to adopt a "go slow" approach, in which each necessary modification could be studied in depth, if necessary, to determine whether APL, or the operating system, or both, needed to be changed. Since no one from Sharp would be familiar with our operating system, our staff would have to be involved with the APL code, regardless of who was actually making the changes in APL.

Secondly, there were a number of local "enhancements" which had to be installed in the Sharp software, using code taken from the existing APL system. Clearly, a knowledge of the internals of the APL system is necessary if it is going to be modified. Since we were going to be doing the "enhancing", we needed to know as much as possible about the APL internals. One way of doing this would be to do the installation, by virtue of which we would be forced into some contact with the APL code.

Lastly, since the Sharp system at SU is unique in the sense that it contains code not run elsewhere in the world, a great deal of the burden of maintaining it would logically fall on us, with Sharp providing backup support for bugs which we determined were not in any of the code we had added or modified. But, in order to arrive at a decision as to whether our code was at fault in any given situation, we would have to have the capability to analyze dumps, and therefore we would need a certain familiarity with the internals of the APL system.

All of these considerations led us to the decision to be much involved in the installation of APL. Although this probably slowed down the conversion timetable, it has now left us with a considerable ability to deal with APL at the systems level.

Conversion History

Nearly 18 months were required to convert to SHARP APL, from initial planning (October 1978), to an "all fires on low" condition (March 1980). While no one worked full time on the conversion for eighteen months, at times as many as a dozen or so people were working on the project simultaneously.

Based upon various memos, shipping receipts, bills for internal computer time, etc., we have roughly reconstructed the schedule of the conversion.

> October 1978 to March 1979
>
> Pressures for obtaining a new APL system began to build during the summer of 1978. One of us (Cartwright) published a memo detailing the features of the existing APL system which would probably give trouble in any APL upgrade. The "memo" ran to 17 typewritten pages. A series of meetings ensued, some internal, some with faculty, and finally a half dozen or so with vendor representatives.
>
> April 1979
>
> During the meetings, the list of likely problem areas had been refined, but there was still disagreement as to the extent and/or existence of some of them. Thus, we undertook a survey of about 100 randomly selected workspaces. Lists of workspaces were assigned to various staff members for direct examination.
>
> On May 1, Sharp was selected as the vendor of choice.
>
> May 1979 to July 1979
>
> Because of the special operating system run at Syracuse University, Sharp could not supply an APL system which we could run unmodified. Thus, about two months had to be devoted to modifying their "closest" system to run. Some changes were also made to our operating system and telecommunications software to accomodate SHARP APL. On 19 July 1979, APL was running "out of the box".

August 1979 to November 1979

Some features of the old APL system were so vital to our operation that they had to be incorporated into SHARP APL. This effort involved writing code both in APL and in assembler. The largest examples: an accounting system, an Internal Job Reader (whereby APL users submit APL vectors as "cards" to the batch operating system), and a workspace and file archiving system (which permit APL users to store and retrieve their workspaces and files on magnetic tape).

Beginning on the 24th of November, SHARP APL began running locally for two hours each morning for testing. The existing APL system continued to be the production system.

June 1979 to January 1980

A development group from Sharp, working in Toronto and Rochester, devised several clever conversion aids which we eventually used to actually move workspaces and files to the Sharp system when it went into production. Testing of the conversion software and numerous dry runs continued up to the moment of conversion.

The "old" APL system was shut down at 5 pm on Friday, 18 January 1980, and the conversion began. During Saturday and Sunday, the accounting data, workspaces, and user files were moved/converted to the SHARP APL system. At 8 am on Monday, 21 January 1980, production operation of our SHARP APL began.

February 1980

The first day we ran SHARP APL at Syracuse was generally a disaster for APL users. We had never tested the system under a heavy load, which turned out to have been a major mistake. The accounting system could not keep up with users signing on and off. Response time to trivial keyboard inputs was more easily measured in fractions of minutes rather than seconds, etc. The Internal Job Reader had to be rewritten (too much had been written in APL, not enough in Assembler, leading to gross overusage of the CPU).

Essentially, most of the problems had been solved by the end of January, and all but two intractable ones were fixed by early March. One of these caused the system to crash several times each week. It turned out to be a timing problem involving the file system, which occurred only when running under our operating system.

Summary

As we look back over the preceding year, we can say that the conversion went about as well as one could expect. Most of the serious problems had been anticipated and resolved on schedule. There were only a few bad moments for any of our users, and very few suffered any difficulties. Most people were quite satisfied with the new system after the first several weeks. Our batch users were inconvenienced by operating system crashes only during early stages of the conversion. Later, there was clearly a burden on machine resources when we ran two APL systems simultaneously. While we worked hard indeed on the conversion, our own staff was able to maintain regular daily work schedules with no extended hours, until the actual conversion weekend.

Epilogue

Now that we have run SHARP APL for nearly six months, we can make some observations about it. Overall, it uses more CPU than our earlier APL system. File sizes and reservation limits of accounts must be much larger to handle the same kinds of data. Disk space used by APL workspaces has increased, due partly to increased workspace size and partly to the impact of *CONTINUE* workspaces. The most important observation to us is that our APL users are genuinely delighted with the new features of SHARP APL.

INTERNATIONAL NETWORKS ENHANCE SPARES SUPPORT LOGISTICS

Wayne Porter Keyes
Amdahl Corporation
Sunnyvale, California

The current need for highly responsive Spares Support Logistics systems is to reduce, control and manage logistics delay time. An on-line, real-time, interactive, user-oriented inventory tracking system which is flexible and adaptive is essential for adequate responsiveness. In an international spares support logistics system designed to maintain overall operational availability above 99% for the entire worldwide installed equipment base, a personal communication element must be included. Only then can the manifestations of indeterminacy, inherent to logistics, be controlled and managed.

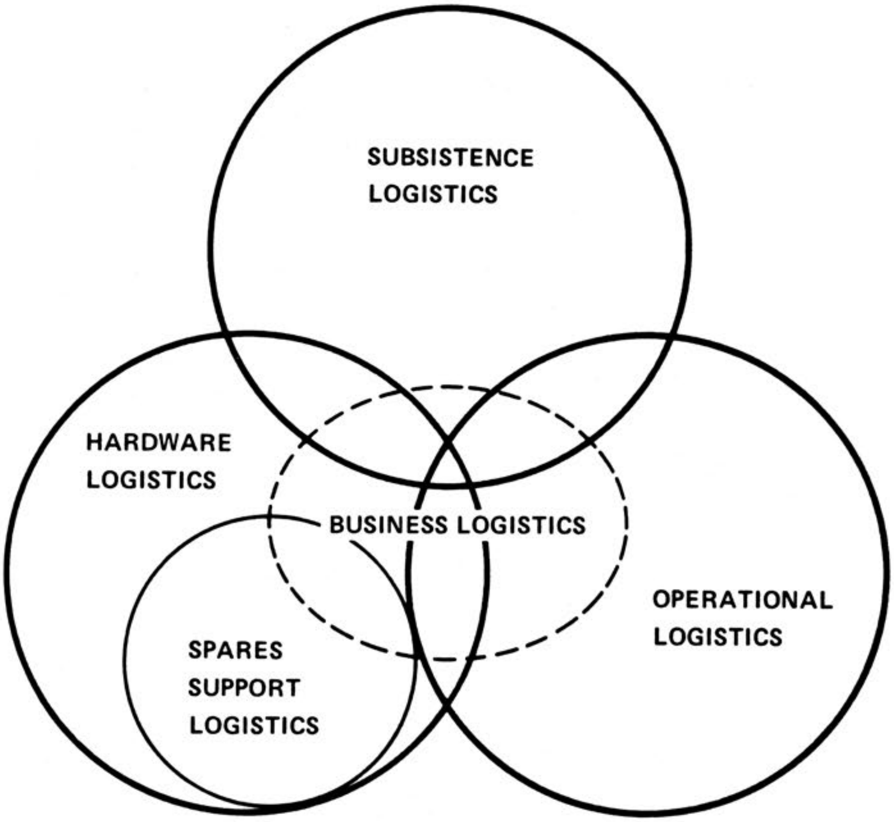
Logistics and Its Sub-disciplines

It is my contention that the significance of the above assertions becomes more apparent once we agree upon the definitions and roles of **Spares Support Logistics** and **Logistics Delay Time**. The importance of these concepts extends from early business planning to ultimate customer (hardware) support, corroborating my premise that **Logistics** is a systems management discipline deriving from business and engineering.

In a national university short course consortium, the University of California at Los Angeles and the Virginia Polytechnical Institute and State University offer a course entitled "Integrated Logistics Support (ILS): Elements and Application". Clinton Van Pelt, the course coordinator (and lecturer), divides Logistics into three sub-disciplines.

"**Subsistence Logistics** includes the basic necessities of life: food, clothing, and shelter. In a given environment, these needs are constant and predictable. We know what we need, where we need it, and the results of not having it. Subsistence Logistics is essential in industrial societies and is the principal activity in primitive societies. In the industrial societies, Subsistence Logistics is dependent on Operations Logistics."

"**Operations Logistics** includes the systems that produce the necessities as well as the niceties of life and the raw materials these installations process, the fuel to energize the installations and the operators of the installations. Operations Logistics is also constant and predictable. Steel makers, once they know how much steel is to be produced, know with a high degree of precision the amount of fuel, the amount of iron ore, the number of blast furnaces, and the number of operators required to produce the specified amount of steel. They don't know, however, when a blast furnace is going to break down, what is going to be required to repair it, or how long it will take to effect the repair. Operations Logistics is dependent on Hardware Logistics."



Subdisciplines of Logistics

Figure 1

"**Hardware Logistics** consists of the resources required to keep systems in operating condition. These resources are: spare parts, publications, test and support equipment, personnel and training, training equipment and facilities. These resources are the true subjects of the integration process. Hardware Logistics requires a specific integration activity because:

> Succeeding generations of hardware continue to incorporate technological advances which require revised approaches to their support. Subsistence Logistics and Operations Logistics have required little revision in their approaches to supplying the physical needs of man or the operational needs of equipment.
>
> For a given operational scenario, the demands for Hardware Logistics are random while the demands of Subsistence and Operations Logistics are steady state."

It can readily be seen that **Spares Support Logistics** may be defined as a subset of Hardware Logistics. Spares Support Logistics includes the functions of planning, provisioning, procurement, the traffic and transportation of distribution, warehousing, inventory control, inventory management, repair, modification and retirement. It impacts personnel, training and technical publications. Revenue generating spares distributed and warehoused solely for resale do not satisfy the concept of Product Support as used herein and, therefore, are excluded from this special subset of Hardware Logistics. The special elements of Product Support pertinent to the Logistics of Spares Support are:

1. Technical skills availability and mobility;
2. Spares availability and mobility;
3. Component repair and the Repair Pipeline;
4. Engineering changes and the modernization of customer equipment and the spares inventory.

Relationship of Logistics Delay Time to Operational Availability

The current need in Spares Support Logistics is to reduce, control, and manage Logistics Delay Time (LDT). No systematic, sophisticated, specific attack has been directed toward the reduction of Logistics Delay Time as has been directed toward the improvement of reliability and maintainability. This is most likely because improvements in reliability and maintainability not only were more readily attainable (both being amenable to operations research techniques) but also because such improvements were more readily confirmable and measurable. These latter attributes not only enhance the attack by the designers upon reliability and maintainability, but also focussed the attention of the customer upon the results of the designers' efforts.

As systems become more sophisticated, more capable, and more costly, the system value (or, cost effectiveness) becomes more significantly a function of the system's operational availability (A_0). Consider:

$$A_0 = \frac{MTBF}{MTBF + MTTR + ADT + LDT}$$

where: MTBF is the system reliability expressed (usually in hours) as the Mean-Time-Between-Failures;

MTTR is the system maintainability expressed (usually in hours) as the Mean-Time-To-Repair;

ADT is Administrative Delay Time. This is manageable within an organization, but then remains either relatively fixed or a relatively constant function of the extent of action required;

LDT is Logistics Delay Time. It is the measure (usually in hours) of the elapsed time from identification of the required replacement part to restore a system's availability, until the delivery of that part to the point of need in the system.

As achievable differential change in reliability and maintainability decreases, the significance of any reduction in logistics delay time increases. In the successful quest for operational availability above 99%, reliability has reached a nearly asymptotic state. Reliability increases (while still sought) can be achieved only via technological breakthrough resulting from tedious and expensive research and development programs. Maintainability has been receiving increasing attention over the last decade. Design management has become aware that maintainability, like reliability, is irrevocably a function of the design process ultimately establishing the cost of maintaining a system in operation. Hence, maintainability is approaching an asymptotic state where differential improvements will become increasingly more expensive.

It is not enough to recognize that reliability and maintainability are quantifiable, stochastic domains regularly established in the engineering sciences. They are that, of course, but indeed, they are much more. First, in review of the foregoing, they are both a function of the design process, and they both can be readily defined, structured, and measured in a sophisticated, scientific manner. Beyond that, there are established techniques for allocation and demonstration in addition to assessment. Design trade-off decision processes are well established and routinely practised.

Logistics delay time is far removed from that level of sophistication and is far from a comparable level of performance. Significant improvements are, indeed, attainable at fully acceptable costs. However, the improvements are no longer "easy" to obtain. There must be precise planning, organization and control methods energetically prosecuted to achieve meaningful reduction in logistics delay time. However, such management of logistics delay time can improve system availability, increase manpower effectiveness, decrease investment in inventory, and enhance customer satisfaction. Hence, the increasingly significant costs (mostly in spares inventory, manpower, premium communication, and premium transportation) necessary to reduce logistics delay time are more than offset by the combined tangible and intangible increases in systems effectiveness.

Another way of phrasing the foregoing is that thc significance of logistics delay time increases as operational availability (A_0) approaches unity. Increasingly greater attention will be given to broader distribution of spares in greater range and depth; hence investment, depreciation, and taxes will all tend to increase rapidly. Not only will demand for premium transportation increase (hence, cost), but also innovative employment of those services will be required. But premium transportation by itself is insufficient as the requirement for assured transportation becomes essential. Assured communication is equally important, and communication time becomes a significant factor. The element of customer satisfaction becomes influential in both immediate and medium range decisions. While costs and times can be measured, assurance and satisfaction are subjective, if not indeterminant, and certainly difficult to apply to orderly decision techniques.

As logistics delay time becomes smaller, further decreases become increasingly more difficult and more costly to achieve, and management requirements become more stringent. Concurrently, the percentage of in-stock spares becomes more important. In system down situations, after determination that the problem is hardware caused, diagnostics may initially isolate the probabilities to certain "blocks" of field replaceable units (FRUs). An available option is to immediately move these blocks to the requiring location so that the logistics delay time parallels the remaining diagnostic time. Note that the benefits of such techniques for reduction of the total maintenance down time for the situation at hand generates a complication to stock management and communication for subsequent readiness. Good, unused spares are temporarily away from their normal storage location. The result of this complication is a risk of increased logistics delay time to a subsequent down system.

In system down situations requiring the movement of spares, transportation mode selection is a logistics delay time variable, a cost variable, and an assurance variable. The factors that significantly play on this decision include the time of day, the day of the week, the to and from locations, the distance involved, sometimes the terrain or the weather, inter alia.

The Logistics of Spares Support

Obviously, the minimum logistics delay time would result from elimination of any requirement for the movement of spares. Equally obvious, however, is the usual infeasibility of the placement of spares of all field replaceable units at all system sites. The selection of site spares, then, is a process of utmost importance to operational availability. Careful application of statistical techniques for spares provisioning is fundamental to site spares selection. Statistical techniques, however, require data input which are seldom better than educated, judgemental predictions until well into the employed life of uniquely designed hardware. This indeterminacy will be manifested by instances of initial procurement of inappropriate spares quantities or, perhaps, inappropriate placement of these spares. The value of data collection for the reduction of indeterminacy in reliability and maintainability assessment is well recognized. A partial intent of this paper is to invite attention to the role of this same indeterminacy in logistics delays. Toward this end, Table 1 is submitted to reflect on the variables employed in spares requirements calculations.

Where the effect of the error is noted as impacting spares quantities, it may also impact the spares placement decision or strategy. In either case, if the error is toward fewer spares, toward limited dispersal, or toward increased repair workload (i.e., more spares in the repair pipeline) such error will usually be manifested by occasions of extended logistics delay time. At a minimum, the management and control of the manifestations of these indeterminacies will require a highly flexible, adaptive inventory control system which includes a rapid, reactive, assured communication system.

In the process of site spares selection there are two predominant, competing factors: cost, on the one hand, and percentage of expected demands, on the other. Obviously, spares that are selected to be placed at each system site will require the greatest quantities in the worldwide inventory (i.e., the greatest investment). On the other side of the ledger, site spares will provide minimum logistics delay time. If the analysis is astutely performed, the selected site spares will provide replacements for the great majority of system hardware failures (80% is considered achievable). The selection of site spares, then, is that portion of the overall logistics analysis demanding the greatest rigor. The result represents an investment for customer satisfaction.

If, indeed, spares of 100% of the field replaceable units (FRUs) are not to be placed at every site, then where are they to be placed? The answer to that question formulates the spares support strategy for the worldwide installed base of systems. Decisions deriving from this strategy include categories of storage depots, number of each category, and the analytic methods to be used to determine the spares stock to be placed at each depot of each category (techniques, models, decision elements, and source). The location of each depot should be the subject of careful analysis. Transit time to sites served by each depot should be traded off against the costs of the depot, transportation, and communication. However, each depot represents an investment for customer satisfaction, and this subjective element must be considered in the decision process.

Variable	Source	Determinacy	Effect of Error
Number of Systems	Marketing Forecast	Prediction	Spares Quantities Repair Workload
Geographical Density	Marketing Forecast	Prediction	Spares Quantities Spares Mobility
Item MTBF	Reliability Estimates	Prediction	Spares Quantities Repair Workload
QPEA	Design Drawings	Determinate	Incorrect quantity calculation
Recoverability	Maintainability Estimates	Prediction	Spares Quantities
Procurement Lead time	Vendor or Manufacturing	Influenced by Maturity	Replenishment Quantities
RTAT	Maintainability Estimates	Prediction	Spares Quantities Repair Workload
Required EC Rework	Engineering Estimates	Prediction	Spares Quantities Rework Workload
Obsolescence	Technology Predictions	Prediction	Spares Quantities Inventory impact

Analysis of Manifestations of Indeterminacy

Table 1

MTBF = Mean-Time-Between-Failure
QPEA = Quantity per End Article
RTAT = Repair Turn-Around Time
EC = Engineering Change

The decision to locate spares in depots is usually influenced by one of two major considerations. Those field replaceable units expected to have very high reliability (consequently have very low expected incidence of failure) will require only small quantities as insurance items. Those field replaceable units having a very high cost (or otherwise classified as a scarce resource) will need careful control of minimum assets.

The increasing importance of decreasing time differentials requires pre-planning for emergency decisions. "Cookbook" solutions may suffice for many of the more common situations. However, the general attempts to minimize logistics delay time create complex situations which, during emergencies, require new applications of old decision variables. The noteworthy factors here are transport mode, alternate source selection, and special pick-up or delivery. The principal variables are time, assurance, and cost, all of which can be studied during pre-planning. If international boundaries are involved, pre-arranged relationships with customs brokers are essential.

Spares support logistics creates many new effects upon inventory management decisions. A typical inventory management practice is to purge items which reflect no (or very low) demand. In a spares inventory not only must insurance quantities be maintained, but also logistics delay time remains a viable variable in the location decision. There is a trade-off decision to be made relative to the acceptance of risk while items are in the repair or rework pipeline versus the procurement of more hardware. However, the most important decision requiring the highest level management consideration involves the planning strategy of making the spares support logistics system either hardware or management intensive. Minimized hardware inventory requires increased inventory management. Minimized inventory management requires increased hardware assets. Minimizing both assures failure of the spares support logistics system.

The Amdahl Approach

Consistent with the foregoing discussion, Amdahl Corporation has developed a dynamic spares support logistics strategy for maintenance of the operational availability at over 99% for its 470V series of computer main-frames. The significance and success of this strategy is indicated in the results of a recent survey conducted by Datapro Research Corporation and reported in **Computerworld**, May 19, 1980. This survey revealed customer satisfaction among owners of the top ten computer mainframes rates Amdahl number one.

The inventory of spares is strategically located at four stocking levels (with some stock "tailoring" to provide for certain unique requirements for Europe, Canada, or the United States). These four levels are site, Metro depot, major item depot, and distribution center.

Site spares are selected by classic ABC analysis to provide replacements for more than 80% of expected hardware failures. In addition to essential small hardware (air filters, fuses, panel lamp bulbs, etc.), the site spares are primarily Amdahl basic logic cards, fully populated and wired printed circuit boards for Original Equipment Manufacturer (O.E.M.) console units, and fans (Amdahl computers are air cooled).

The Metro depots are populated with spares to service either of two functions. The first function of the Metro depot is to hold inventory of insurance type items having moderate cost and expected high reliability. By the placement of a Metro depot within no more than an hour transit time from every site, logistics delay time is kept small when supporting emergency demands for this type of hardware. These items provide replacement for another 8 to 9 percent of expected hardware failures. These Metro depot spares include O.E.M power supplies, cathode ray tube assemblies, Amdahl basic logic cards (beyond those few types stocked on-site) and blower assemblies. The second function of the Metro depot is to hold inventory for the immediate replenishment of site spares when an item from site spares is installed to replace a defective system component.

The "major item depots" are where most of the "tailoring" is done. Two principal types of hardware are held in these locations. The multi-chip carriers (MCCs) are the Amdahl designed application of large scale integration (LSI) technology which allow the Amdahl 470V series computers to be air cooled. There are about 110 varieties of these plug-in MCCs used in the various models of the 470V. The second type of hardware held in these depots is the O.E.M console drawer assemblies. Both types of hardware are very high cost items, but collectively they provide replacements for approximately 11 to 12 percent of expected hardware failures. Due to the high cost of these items, a considerable customs duty exposure requires special Canadian attention and even greater management in Europe. These depots are referred to as "Country depots" in Europe. The placement of one such depot in each country allows the European Logistics Manager to more readily address unique customs regulations. However, where the density of systems so dictates, additional sets of MCCs are placed at other locations within a country. In Canada, where the unique problem is the support of systems across vast geographic expanse, the country is divided into three regions, and one such depot allowance is allocated to each region. In the United States, this hardware is placed in 15 depots operated, under contract, by a major air-freight corporation acting in fiduciary capacity to Amdahl. The three distribution centers are located in Sunnyvale, California; Toronto, Ontario; and Schipol, Nederland (a suburb of Amsterdam). The primary purpose of the distribution centers is to provide immediate replenishment to both Metro and major item depots when items are used upon system failure. A secondary purpose is a repository for certain insurance items whose failure is considered to be a rare event, but one that would disable a system.

The original Spares Inventory System (S.I.S.), still in use for financial inventory accountability, uses hard-copy-to-keypunch input to batch update with off-line report generation. As the worldwide installed system base became significant, the inadequacy of S.I.S. as a spares support logistics system became more apparent. In the world of low logistics delay time requirements, it was hopelessly slow. Reports required manual update entries to stay current with emergency requirements. It could not provide inventory tracking assistance to Canada and Europe as it was not on-line. This latter inadequacy plus the implementation of the Metro depot stocking strategy necessitated an on-line, real-time, interactive, user-oriented inventory tracking system. Just such a system, for Canadian spares tracking, had been designed by an Amdahl software engineering specialist in Montreal, Quebec, using SHARP APL.

The Inventory Tracking System: "888 AMDAHL"

The system architect had never foreseen the extension of his "888 AMDAHL" system into a worldwide inventory tracking system complete with repair program tracking capability and replenishment ordering and status. Considerable modification and extension of the program became necessary to bring this about. The system architect spent two weeks in the Sunnyvale, California, headquarters of Amdahl Corporation to analyse the total logistics system, to plan the modifications, and to initiate the program revisions. Subsequent program control was effected from Montreal. The results were rapid and ingenious. The data base, reflecting the basic inventory, was flexibly restructured so that the various generations of the same part evolving via engineering changes could be either uniquely or collectively confronted. The location hierarchy could be listed separately or collectively. All of the usual inventory change functions were provided, including both in-transit and broken-in-transit status. The job of the data entry operator was enhanced by a series of very clear prompts (sometimes delightfully revealing the French-Canadian cultural background of the architect). However, the system origin as a simple inventory tracking tool spawned one short coming. It had been designed for simple decrement of inventory to transit or increment of inventory from transit on SEND or RECEIVE functions, respectively. No real-time memory was provided to enable transaction reporting. Hence, the need for electronic mailbox became obvious.

The architect designed a mailbox into "888 AMDAHL" that was separate from the I.P. Sharp "666 BOX". His intent was to force message retrieval by the recipient upon log-on, thereby adding to communication assurance. This probably could have been accomplished by internal program functions using "666 BOX", but an unforeseen benefit has evolved in a recent program extension which will be mentioned later.

The initial program did not have a SEND ORDER function nor a back-order file. When these were added, no capability was provided to edit an order once entered. The INSTALL function (used when parts from inventory are installed in a system to replace a machine part) does not provide for a change in the suffix to the part number (indicative of engineering change level). There still is no means to compare the actual inventory of a location against an allowance list to reveal exceptions (excesses or shortages). While none of these deficiencies is lethal, they are indicative of system development by a purely software oriented architect when he is physically removed from the logisticians.

Flexibility and Adaptability for Spares Support Logisitics

The spares support logistics system that was described under "The Amdahl Approach" and "The Inventory Tracking: 888 AMDAHL" appears almost simple in its orderliness. Indeed, it might be if hardware failures would occur in orderly, predictable patterns. However, Murphy's law is especially active in the world of logistics. Logisticians believe that a corollary to Murphy's law is that the item most likely to fail is the item that is in most critically short supply in the spares inventories. A failure filled from a European country stock when the Schipol Distribution Center is already zero balanced cannot be the subject of a routine replenishment order. The pain level here is not only derived from the transit time, Sunnyvale to Schipol, (assuming parts availability in Sunnyvale) but also from customs clearance requirements, sometimes an impossible barrier on weekends. Mailbox has aided critical communications.

If a defective item is registered by serial number prior to export back to the United States for repair, that same item may be returned with customs dutiable only on the repair costs. This is referred to as "value added" and is usually considered to be 10% of the original item value. This requires careful serial number tracking from the time of export, in transit, through repair, and return. While "888 AMDAHL" provides some facility for this tracking, mailbox use is mandatory to alert the right people for timely action.

The stock levels, hour-to-hour, in the air-freight depots in the United States are likewise essential to the management of logistics delay time. An APL terminal has been placed in the air freight coordination center for most rapid entry of their unique function, "EMERGENCY PARTS". This is the unexpected benefit recently derived from the internal mailbox. Each time the EMERGENCY PARTS function is exercised (which is about five to ten times daily), a mailbox message is generated to several key Amdahl offices detailing the move. The necessary flexibility and adaptivity was available in the system. This air freight depot form of emergency parts logistics would comprise the subject of yet another complete paper.

Conclusion

These examples of adequate responsiveness deriving from a flexible, adaptive, on-line system are offered in evidence of how international networks enhance spares support logistics. In the discussion of the logistics of spares support, the manifestations of indeterminacy were shown to be inherent to logistics. The personal communication element provided by electronic mail (and augmented by telephone) allows control and management of this indeterminacy and its manifestations.

TEACHING APL IN AN ACADEMIC ENVIRONMENT

Gillian Wade
Institute of Computer Science
University of Guelph
Guelph, Ontario

Introduction

The unfortunate division between scientists and non-scientists first explored so perceptively by C.P. Snow in his seminal work **The Two Cultures** (in 1959) has not appreciably narrowed in the intervening years; rather, the continuing technological revolution has, if anything, exacerbated this seemingly unbridgeable chasm. The paradox, of course, is that the mounting wave of technological achievement is largely due to specialization — in universities and industries, research labs and businesses — and, in turn, its maintenance and further development demands an even higher degree of specialization. One of the most important and frustrating gaps today is between those people (and not all of them are scientists, by any means) who feel comfortably at home with computers and those who don't.

As computers play an increasingly important role in all our lives (which critics, unfamiliar with these machines, darkly hint is an ominous intrusion), it becomes imperative that we strive to dismantle the barriers that exist and neutralize the growing polarization between "**them**" and "**us**", and work towards a more widespread acceptance of the computer as another powerful and highly useful tool, which should not be beyond the grasp of any reasonably intelligent person to use.

Those of us who are already involved in this exciting and ever-expanding field must remember, as one of my students pointed out in his feedback comments, that there is "life beyond computing", and it is up to us, to all of us, to reach out beyond our ivory towers and computing centres and spread the knowledge and share the experience we have gained. We have an obligation to our non-computing colleagues to do our best to dispel the modern myth of the computer as a mysterious and infinitely complex "black box with flashing lights" that can only be used by a highly-trained and specialized elite. Computers are for everyone to share! Indeed, the rapid growth of the home-based microcomputer industry is hopefully an early clue to the new direction, but there is still much for us to do.

For many years, computer professionals have too often been their own worst enemies in forging the negative image so many people have of computers, for everything they did was couched in highly technical jargon and complicated buzz-phrases. Simply to use the machines required more than a passing familiarity with what made them run, and an extensive knowledge of the intricate computerese of COBOL, FORTRAN, ALGOL and so on... but there is no longer any excuse for this situation, for now we have an ideal language for non-computer people: APL.

APL at Guelph

The joy of APL is that it allows so many people, with so many diverse requirements, to use the resources made available through the computer with a minimum of technical know-how. The comparatively easy acceptance of APL is as apparent in the commercial and industrial world as it is in the academic environment with which I am most familiar — perhaps even more so, for unfortunately, many computing science professors still seem to regard APL as a rather fun toy but not the first choice as a language one would select for any serious work.

One of the most gratifying aspects of my own role at the University of Guelph is the opportunity I have to introduce young school-children to the world of computers. Each year, several special groups, many with children as young as the age of ten, come to the Institute of Computer Science for an introductory course in APL. These young people have no pre-conceived notions or fears of the machine and within hours of their first acquaintance with the system they are happily using APL at their terminals with far greater ease than I remember my own generation acquiring the three "R's", which is a promising indication of future trends.

Older students — and I don't just mean those enrolled in university undergraduate courses — are a somewhat different matter, and the manner in which these people are introduced to computers in general, and APL in particular, is of great importance. The wrong approach can even aggravate rather than overcome the latent suspicion and even dread of computers that, as I have remarked, is still so widespread among those with little or no computing experience. There are probably as many ways to teach APL as there are rival theories for the teaching of conventional foreign languages, but I will outline here what we are doing with marked success at the University of Guelph, and hopefully you will find at least parts of our approach to be useful in your own situation if you are faced with the job of arranging an introductory APL course for students, employees or clients.

At Guelph, degree courses in computing are offered by the Department of Computing and Information Science, but despite the dramatic growth in the use of time-sharing on campus over the past few years they still have no credit course where students can learn APL. The procedural languages have traditionally been dominant for it is felt that they give students a better understanding of the technological sub-structure. One or two of the introductory lectures, as well as courses offered by the Department of Mathematics and Statistics, do include a brief discussion of APL. But as one leading faculty member confessed to me, the primary reason that this situation has not changed is because of bureaucratic inertia.

However, students in such diverse disciplines as Economics, Zoology and Hotel Administration are often required to use APL for some of their assignments; in addition, an increasingly large group among the faculty and administrative staff are discovering the convenience of APL to help with their research or data processing. To meet this demand, all formal APL training has become the responsibility of the Institute of Computer Science, the organization which handles the bulk of computing services for the campus.

The institute offers a regular non-credit introductory course in APL, as well as many ad-hoc seminars to inform users of new features, tricks and techniques we've developed, new public library programs, and more sophisticated APL topics. Some people, of course, have managed to learn APL on their own, with just a little help from Institute staff, as a means of solving problems in their particular discipline, but the majority of our users have at some time taken our introductory course. I'd like to outline now how our course is organized, and why we have chosen to do it in this particular way.

Our APL Teaching Program

Our first course in APL was offered in September of 1970, at which time some 30 students were introduced to this versatile language. A decade later, we are teaching well over 300 people a year, and indications are that demand will continue to be high. In this ten-year period the course has gone through several changes. We have now arrived at a format which we feel best suits the needs of our users, though of course we are constantly refining the content in line with the evolution of the language itself.

Obviously, some of the considerations we have had to bear in mind won't necessarily apply in your case. For example, our courses are taken by an extremely diverse group of people, including undergraduates, graduate students, faculty members, researchers, administrators and technicians. Their computer expertise ranges from none at all to several years experience with some of the more traditional languages and systems. A company organizing an APL course for some of its employees, for instance, can expect a more uniform group of students. We are also fortunate in that all the individuals we teach are highly self-motivated. Some of them merely want to acquire a minimum amount of computer expertise to apply to their own field of interest, and have chosen APL as the most convenient tool, while others are interested in extending their computer skills and becoming APL programmers themselves.

It is most important to first analyze your potential audience in order to achieve the right level of presentation. It's very easy to oversimplify and spend a disproportionate amount of time on what may be a trivial topic; and it's equally easy to take for granted concepts you have already mastered but which may be more difficult for a novice to grasp at the first introduction, skimming over points which may deserve more detailed explanation. Once you have lost your students' attention, whether from oversimplification or overcomplexity, it can be hard to regain it. The dividing line is not at all clear, nor is it always possible to achieve, but it does make the task easier if some initial assessment of this kind can be made.

Another factor that has influenced the presentation of Guelph's APL course is the available technical facilities and teaching personnel. Because of physical limitations, and because we have found that a smaller group is more rewarding both for the teacher and the students, we have an enrollment ceiling in each of our classes. At the moment this ceiling is 30 people, which we feel is still too high, but because of resource constraints we are unable to lower it. The ideal arrangement, perhaps, is to have every student seated at his or her own terminal so that they can immediately try out each new concept as it is presented by the instructor. Unfortunately, we are far from achieving this one-to-one goal, and so we have had to adapt to what is available. There is a CRT in the classroom for the instructor's use, and our computing laboratory is equipped with 26 terminals for everyone to use. In addition, several public terminal pools and departmental machines ensure that no one has difficulty in getting access to the computer at times of their own choosing to try their hand at APL.

The demand for places is high, and so each semester (that is, three times a year), we teach at least three separate courses. Indeed, last summer overflow registrations necessitated the scheduling of two additional sessions. The timing is important, since many of our students need to learn their APL early in the semester in order to apply it to their other work. So we arrange overlapping rather than consecutive courses, and each session is taught by a different staff member, none of whom, incidentally, is a professional lecturer.

It was decided early on that uniformity in presentation was important, since we wanted to ensure that students taking any of the sessions all covered the same material, and we wanted to make it easy for a student who was unable to attend, say, the second class of his particular group to sit in on the same lecture with another group at another time without missing anything. Obviously, the personal enthusiasm and knowledge of the individual instructors will have a decisive influence on his teaching style and his audience's appreciation of the subject, but by providing a uniform framework for the course we believe we can supply our students with the best instruction while at the same time making it easier for newcomers to our APL team to teach the same course by giving them the benefit of our cumulative experience. Everyone uses a common set of notes, slides and assignments, which naturally reduces time spent on class preparation. Introductory manuals on APL and the file subsystem are given to all attendees. There is no required textbook, though we are frequently asked if there is one available. We recommend "APL: An Interactive Approach" by Gilman and Rose, and Paul Berry's "SHARP APL Reference Manual" is also becoming popular.

Originally, three separate mini-courses were offered, but since most people took all three anyway, it seemed more sensible to amalgamate them into a single introductory course. This course is divided into six modules, each three hours long, given twice a week over a period of three weeks. So students receive a total of 18 hours of formal instruction, 12 in the classroom and 6 hours of supervised practice in the laboratory. This format is well suited to the needs of our students, and again, our experience has indicated that dividing the material into smaller "chunks" with breathing space for practice in between makes it easier for them to learn than presenting a concentrated course jammed into a couple of days. The threshold of "mental indigestion" is fairly low, and though crash courses can be effective, we do not feel this approach is appropriate for our purposes at the introductory level. We have recently experimented with spreading the course over a six-week period, but the consensus is that a seven-day gap between classes is too long, and the present scheduling seems to be the most effective.

During the two-hour classroom period the instructor demonstrates each point using a terminal hooked into TV monitors, and students are encouraged to ask questions and discuss various points at any time. Immediately after the lecture there is a one-hour lab session in which students can take turns at the terminals to get some "hands-on" experience while the teacher is present to give answers and deal with problems or misunderstandings. Some simple assignments are given out and the students are expected to spend time at the terminal on their own to try out what has been covered in class, for only when the theory is put into practice is it possible to see if the concepts were understandably presented. Any difficulties or "fuzzy" areas are discussed at the beginning of the next session. Because of the students' differing interests it isn't possible to tailor assignments specifically to everyone's needs, and so instead we choose exercises and examples from the standard textbooks. 81% of the students complete most of the assignments, and of those unable to do them all, 88% said they were unable to find the time. This is an insoluble problem, however, for many were still unable to devote more time even when the course was spread out over six weeks.

The six modules have been constructed so that each builds on the others, forming a continuous whole, while at the same time being sufficiently self-contained to allow those people who do not wish to attend the full course to sit in on just those topics they wish to cover either to get started or for revision purposes. The first session introduces the concept of time-sharing in general and the basic elements of the APL language, including: monadic and dyadic scalar functions; the concept of scalars, vectors and matrices; assignment; selection using indexing, take and drop; the reduction and scan operators; and rho.

In the second class, students learn workspace management, the more frequently used system commands, system functions and system variables, and the APL public library structure. For some, this is as much as they will need. Some 70% of the students, however, proceed to the third module which covers more of the language including the relational and logical primitives, character data, compression and expansion, selection on matrices, inner and outer products, and a variety of other array manipulations. The fourth and fifth sessions cover writing, editing and debugging programs, branching, mixed output, quad and quote-quad. Students are given a programming assignment which is demonstrated in class. The final lecture is devoted to a brief overview of the APL file system.

A recent survey of all the Institute's non-credit courses provided some interesting statistics about how readily most people accept APL. 94% of those responding who had taken our APL course stated that their interest in computing had been increased, and 92% now found it easier to use the computer. Nor were these all novices; about half had had other exposure to computers. 87% stated that they had been able to apply their new-found knowledge to their own work. One reseacher lauded APL as giving him a "quantum leap" in his ability to analyze his data.

A look at some of the APL projects now in use or being developed on campus also offers an intriguing glimpse of the rich variety of applications to which the language is being put. APL is used for diet analysis, for the simulation of a hotel's back and front office operation, to teach perspective drawing for landscape architecture, wildlife management, and to maintain personal bibliographies and collections of references. On the administrative side, an APL system maintains all the University's complicated budgeting. A veterinary professor, delighted at the new prospects opened up to him, bought a terminal and a flat-bed plotter and quickly wrote some simple APL programs to produce graphs of equine electrocardiograms and other research data of a sophistication that previously had been almost impossible to achieve. For several years, a group of physicists had been attempting to translate some extremely complex matrix convolutions for analyzing X-ray spectra into FORTRAN. Within three months of taking our introductory APL course, they had successfully produced the results they wanted. True, they spent ten thousand dollars to do it, but they have written several major research papers about their findings, and they now have a system which, as far as we are aware, is the first to apply these APL techniques to atomic physics.

Conclusion

We strive constantly to improve our courses to provide the best training we possibly can, and feedback from our students is important to us. Each person attending is asked to complete a course evaluation form, and these are used to define our areas of weakness and try to improve them. We spend a great deal of time consulting with and giving advice to our students after they have completed the introductory course to help them develop their own APL skills. To provide easy access to the computer, free account numbers with a modest spending ceiling are provided to users who have no other source of funding. As an indication of the wide use of APL, these free numbers accounted for some 30,000 computing dollars in the past fiscal year even at our comparatively low internal rates.

Where do we go from here? A common problem facing all academic institutions in the coming years will be tackling an increasing workload with the same number of personnel, and with even less in some cases, in the face of cutbacks in educational spending. In business too, individual productivity will assume ever greater importance and since this is an area in which APL excels, its use will inevitably continue to grow. At Guelph, we are planning to upgrade our APL self-teaching courses so that the computer itself can play a more prominent role in teaching its own use, and we would like to develop pre-recorded audio-visual presentations, thereby reducing some of the load on our human resources.

As several of the other papers presented here have clearly demonstrated, APL offers many technical advantages over rival computing systems, but I hope this account of our training methods at the University of Guelph has demonstrated yet another virtue of APL, in that its comparative simplicity and flexibility make it the ideal language for teaching computing to those people who need it as another tool for use in their own particular area of interest. And in offering a relatively painless introduction to the enormous benefits of computer resources to people with little or no previous experience, it is building a sturdy bridge across that gap which divides so many specialists in both academic and non-academic environments.

USE OF TELEX FOR FINANCIAL PLANNING & REPORTING

Roger W. Shaw
The Wellcome Foundation Ltd.
London, England

Abstract

For financial planning to be of value, particularly when dealing with short term forecasts, data must be collected, processed, and presented rapidly. Since the end of 1978, The Wellcome Foundation Ltd., a British-owned research-based pharmaceutical company with an annual turnover of 380 million pounds, which operates in most parts of the world, has used direct Telex access to speed the collection and validation of data from approximately 30 countries in all continents. This paper outlines the systems used, describes in sufficient detail to serve as a practical guide the ways in which they were modified for Telex use, and assesses the success of the project and some implications of this approach in respect of reporting organisation and structure.

Introduction

The economic climate of today presents a challenge, especially to international companies such as the Wellcome Foundation Ltd., a British-owned research-based pharmaceutical company operating via subsidiaries and agencies in most parts of the world. To meet this challenge many things are required, among them the provision of accurate and timely information relating both to current performance and future plans, and it is primarily in this area that the Group Corporate Planning Directorate has its activities.

Recognizing the importance of speed and accuracy in the planning process, Wellcome's Group Corporate Planning Directorate embarked on computerisation of its planning systems in 1974. Today the Directorate has several large computer-based systems for different areas of planning, all of them written in APL; it is with one of these systems, known as UNICORN, after the company logo, that this paper is concerned.

An Outline of the UNICORN System

UNICORN is the main planning system used by the Directorate, and was the first to be computerised. In 1974, a first version was implemented using the I.P. Sharp AIDS package, which provided a useful starting point and much experience when, after a year, it was decided to write a customised system specific to Wellcome. Further development has now led to a system which is powerful and sophisticated, yet flexible, comprising some 6000 lines of APL code.

Briefly explained, the UNICORN system enables the user to input financial data relating to various pre-defined time-periods, manipulate that data (in particular by running model programs which calculate and store derived data) and generate reports from it. Naturally there is much more to it than that, such as currency conversion, consolidation, "what-if" capability and so on, but this simple description will suffice for present purposes.

In its earliest form, the system was essentially a budgeting exercise, in which all the relevant entities (overseas subsidiary companies, etc.) completed a set of forms which were then returned to the Group Corporate Planning Directorate, where they were input to the computer. Once in the machine, it was validated, evaluated and used as the raw material for futher exercises. Data was added in respect of intra-group transfers and other non-locally determined items, so that a complete picture of the Group's financial plan over the specified period could be prepared for review by the Board of Directors.

At an early stage, it was realised that considerable benefits would be obtained by making available UNICORN to those overseas subsidiaries who had access to the Sharp network, and whose size would justify their use of it. Accordingly, by the end of 1976, the system was being used via terminals by users in Canada, USA, Belgium, The Netherlands, and Germany as a tool to develop and refine plans before submission. By this time, UNICORN had been extended, and was being used not only for the original annual exercise which looked ahead over 3 years with the budget year being by quarter, but also at quarterly intervals for "rolling forecasts" looking ahead over 18 months by quarter. These quarterly forecasts were basically a simplified version of the full Three Year Plan.

However, because of the volume of data to be processed (approximately 1500 possible lines, covering 8 or 10 time-periods, for each of about 80 Reporting Units), and the process of reconciliation and addition of non-local (i.e., UK-supplied) data, even for the quarterly forecasts it still required up to 6 weeks after the receipt of all overseas data before the final reports, including consolidation to Group level, were ready for presentation to the Board. Allowing for 2 weeks postal delay, this gives a total of around 2 months from the issuing of figures by Reporting Units, to their review. In other words, bearing in mind that the majority of the exercises performed were based on Quarterly Forecasts, it is clear that the better part of one planning period had elapsed by the time any decision resulting from the exercise could be implemented at Reporting Unit level.

In late 1976, with UNICORN well settled down, it was decided to look for ways to shorten this period. Major time-consuming areas which seemed susceptible to improvement were seen to be postal delays, the input of data by Corporate Planning, and the subsequent model running and "cleaning-up" of data (frequently requiring referral back to the Reporting Units), which was all-too-often necessary. None of these problems exist, of course, where the data is prepared on the system by an overseas terminal user — at least, those that do still exist are the responsibility of the overseas user himself, removing a considerable burden from Corporate Planning.

The problem then, was to find some way of providing access to the system to more overseas users. The answer, it was thought, might be Telex. The next problem was that I.P. Sharp had at that time no Telex facility, but nonetheless there were other APL bureaux who had, and Wellcome's interest was so great that a study to determine the feasibility of this approach was initiated forthwith.

Potential Problem Areas

The problems associated with the introduction of Telex access to UNICORN could be broadly grouped into two categories: philosophical or general problems, and technical problems.

The philosophical problems were chiefly:

a) Which Reporting Units were to use the system?

b) Who, within those units, would the actual user (i.e., the person physically at the Telex machine) be?

c) What kind of user training would be required?

d) Could the system be sufficiently simplified and automated?

e) Would the whole exercise be cost-effective?

The technical problems could be summarised in the following list:

a) Would the availability of Telex lines be adequate?

b) Would Telex line quality suffice?

c) What computer service availability would there be? In particular, near 24 hour service would be required, and "Christian" public holidays would need to be covered.

d) Would the Telex line speed of (effectively) 6 c.p.s. be sufficient for the volumes of data envisaged?

e) Could we manage with the very limited Telex character set?

f) Could we manage with a page width of around 58 columns?

g) Could sufficient on-line help be provided?

h) Was the system sufficiently fool-proof — e.g., how might a user cope with the kinds of errors experienced in quad mode (bearing in mind that this was before Event Trapping was available)?

i) How could we reduce "think time" to an acceptable level, for at rates of up to $5 US per minute in Telex charges, this threatened to be the major cost item?

Clearly, some of these questions could only be answered by a realistic trial, and therefore, a somewhat simplified version of UNICORN was implemented on a bureau then offering Telex access in The Netherlands. Although simplified, this trial system had to offer the full range of facilities intended for eventual use if the system were finally implemented on I.P. Sharp. This also gave an opportunity to experiment with alternative ways of achieving some of the UNICORN facilities, such as combining the usual sequence of models into one single model in the hope of a gain in efficiency (in particular with regard to file accesses) as well as simplicity of operation. (In the event, incidentally, the existing modular approach to models was retained).

By March 1978, the trial system was operational, and tests were then conducted from two locations thought likely to be reasonably representative, namely Monaco and Colombia, South America. These tests required a brief visit from a member of Corporate Planning, who went through a complete exercise as it would eventually be run. The results of the tests were very encouraging; in particular Telex line availability and quality appeared quite acceptable. Discussions with I.P. Sharp revealed that their plans for Telex access were well in hand, and as the tests had indicated that the Telex approach should indeed be cost-effective, work on adapting UNICORN was begun immediately.

Modifications to the UNICORN System

The entire process of Forecast preparation had to be examined to look for areas where modifications might be needed. The starting point was to consider the data being collected: should or could this be changed? It was decided that the data items being collected were all essential, but not necessarily for all time-periods. It was decided initially to implement Telex use for the quarterly rolling forecasts, with a view to possibly extending it to the full annual Three Year Plan exercise if it seemed appropriate.

For the quarterly rolling forecasts, the Reporting Units were already divided into "major" and "minor", on the basis of Sales Volume, which broadly corresponds with the overall size of the operation and hence the local effort available. A "major" unit was required to submit a forecast over all time-periods, but a "minor" unit was required only to submit data for the first two, namely Actuals at the end of the last quarter, and Latest Estimates for the current quarter. The remaining time-periods were obtained by interpolation or simple extraction from the most recent Three Year Plan. Inevitably, the direct submission of only the first two columns by "minor" units would open the door to the possibility of problems in grafting the two sets of data together; it was decided that the resolution of any such problems would be the responsibility of the Corporate Planning Directorate after submission. the unit being required only to submit the first two columns in isolation.

The next question was that of forms design. The forms then in use had already evolved into dual purpose documents acceptable to an accountant as an appropriate way of presenting financial information yet capable of use as a punching document. It was felt, however, that an extra column should be added to contain a compulsory "checksum", which would be simply the sum of all the data in the row together with the row (or "ID") number. The fear here was not so much of possible loss or distortion of data over the Telex link, as of punching errors in data preparation by the operator.

Next we turned to the computer system. Up till then, we had been dealing with relatively sophisticated users, and UNICORN acknowledged this by being designed as an "open" workspace, in which the user "rummages around", calling up whichever routines he wants. This is far too error-prone a method for the unsophisticated users we had in mind, and would make the system difficult to learn as well as potentially running up large bills for Telex charges while the user wondered what to do next. Accordingly, it was decided that each user should initially be provided with a CONTINUE workspace which would ensure self-starting (in fact, re-loading of the normal workspace unless suspended in execution of some other function). Routines were also provided, of course, to ensure that when the user ended his session he did so by means of a simulated *)CONTINUE* (i.e., *WALKAWAY*), thus making the process self-perpetuating. On loading the normal workspace, it was therefore necessary to detect whether a user is at a Telex or not: this is a problem which we have solved, though not to our complete satisfaction. One approach, of course, is to keep a list of account numbers of Telex users; this however, is inelegant and error-prone. Another is to identify the IPSA network node number, but this only works if the network is never reconfigured in such a way as to change the number. The most obvious way is to identify the terminal type specifically as a Telex, but unfortunately Sharp have at present no unique type for this, a Telex machine appearing as a bit-paired ASCII terminal. Luckily this type of terminal is actually quite rare, and so this is the mechanism on which we finally settled. It is only fair to admit that so far we have had no major problems as a result of this approach, but it would clearly be much more satisfactory to have a unique code.

User confidence is always important in this type of venture, so the first action the system takes is the printing of a short "reassurance" message, incorporating the name of the country concerned. From here the next step is the automatic selection of the appropriate data file, and of the set of data within that file — a blank set will be set up when the user first accesses the file. The selection of the file is automated by making only the appropriate file "visible" to the user by means of access matrices; at the appropriate point early in each exercise these are set up, and after submission the Telex users are removed, thus guarding against users accidentally (or otherwise) changing their submissions without reference to Corporate Planning. It is worth noting also that, as a consequence of the automatic selection of the set of data within the file, Telex users do not normally have the facility of preparing more than one version of their data. This is not felt to be a serious loss, as it was not anticipated that the system would be used in this way by Telex users, and this has been borne out by subsequent experience.

Having thus painlessly entered the system, the user must make his first decision, as the system asks whether he is submitting 2 columns or ALL, and on giving an answer he comes to the main "menu". This is a very limited set of available commands, in sharp contrast to our customary philosophy of allowing the user free range. In fact there are just 5 options: INPUT, COMPUTE, PRINT, SUBMIT, or OFF. Implementing each of these 5 options involved changing existing methods. Below, each option is discussed in turn, together with the problems posed and solutions provided.

Data Input

The terminal user would usually prepare his forecast "form-by-form", not moving on to the next form until he was happy with the last one. UNICORN therefore contained an independent model for each form. However, this method is relatively time-consuming and calls for a reasonable measure of skill in computer terms. In the interests of simplicity of operation, it was decided that this "piecemeal" approach to the preparation of a forecast would have to be replaced by a "monolithic" approach. Hence it was decided to require that all the data for the basic input lines on all the forms be input before any attempt at modelling can be made.

It was clear that the volumes of data necessitated an approach using paper tape or some other comparable medium. Even with the buffering used by the Sharp system it is inadvisable to attempt to process data as it is received; rather it must be read as rapidly as possible, using in the case of UNICORN direct quote-quad mode rather than an "ask" function, and simply stored until the tape is finished. In practice this means that each physical line of data must be searched for a logical data terminator (in the case of UNICORN the word END is used), and unless this is found the line is merely stored in character form in a buffer within the program and the next one scanned. It is also necessary to use logical line terminators (in this case an oblique "/") since the narrow width of the typical Telex carriage (about 55-60 characters) often prevents a full line of data being contained within a single physical line.

Only when the logical data terminator END has been encountered can the processing of data begin. Apart from any other considerations, there is no means of switching off the Telex machine's paper tape reader under program control (i.e., X-OFF), which means that no error messages can be generated until the tape has finished. At the time of the initial implementation of the Telex facility for UNICORN, Event Trapping did not exist, making exhaustive syntax checking a tedious but absolutely essential process. Once this has been done, and the checksums have been validated and any appropriate error messages have been printed, the remaining successfully read data can be written to file and the user returned to the main "menu". Corrections to faulty data are then effected either by means of another data tape or, if few in number, by directly typing in INPUT.

Model Running

Model running is achieved by selecting the COMPUTE option from the main "menu". As one might expect from the above, the effect is to run **all** the individual models in the correct sequence. One problem which can arise from this is the generation of excessive quantities of error messages, all resulting from a single error in data early in the sequence. The decision must therefore be taken as to whether any of the possible errors which may occur should be treated as "fatal", causing an abort from the sequence of models and a consequent return to the main "menu". After some discussion we decided against this, on the grounds that most data errors would not cause more than one or two messages, and aborting implies losing the possibility of locating other unrelated errors on the same pass.

In many cases, the error messages which could be generated by the models required modification. The two chief causes were that they sometimes used characters outside the Telex set within the text part of the message (e.g., the minus sign in "ERROR 1.3: - NEGATIVE SALES VALUE"), and that the figures causing the error were normally formatted in a manner unsuited to the restricted width of a Telex carriage. Both of these problems were fairly simple to solve, although the latter entailed the sacrifice of aesthetics, for whereas an error message on a terminal giving, say, eight columns of figures would allocate 10 print columns to each numeric column allowing conventional decorations such as commas, parentheses for negatives, etc., on a Telex the decorations are not practicable, and the printed format must have the columns as close as possible to each other. Alignment within columns is preserved as being essential for legibility, but columns are not all necessarily of the same width. Fortunately all model errors in UNICORN are handled by a standard validity-checking routine, which greatly reduced the problems.

In general, some errors are to be expected on the first pass through the data; for brevity and economy the messages are kept reasonably short, but all errors are fully described and explained in the user's documentation. Probably, on receiving an error message, the user will inspect the values of a few IDs using the PRINT option described below, and will then return to INPUT to make corrections before re-running COMPUTE. It may well be that, because of "think" time, this process of correction takes place over several Telex sessions, as it is rarely sensible to work out where the errors lie while still on-line. Eventually, however, the data should be clean, and the user can move on to printing selected lines.

Printing

The user at a conventional terminal generally inspects his data in one of two ways: he either examines selected IDs (perhaps specifying only certain time-periods) or he generates a complete report. There are many standard reports available, the simplest and most often used being "facsimiles" of the original forms. However, because of speed, character set, and page width constraints, the generation of full reports is impractical for a Telex user, so a modified version of the method of inspecting selected lines of data was implemented.

The PRINT option from the main "menu" enables the user to print the values of IDs selectively. The user is requested, on selecting PRINT, to specify a list of IDs; simple means are provided to meet such common requirements as "all the IDs on a particular form", as well as the ability to specify a range of IDs or a straighforward list. One further, very important, option is the reply CHECK. This results in the printing of a list of about 10 "key" IDs, such as Total Sales, Trading Profit, Overdrafts, etc. The particular significance of this option is discussed in the section on submission below.

Overall, it is not usually the case that users perform all their calculation only on the system. The reason for this is mainly cost, as the time taken to print all the calculated IDs would be prohibitive. Instead, the practice is to inspect selected calculated lines, and compare them with those calculated by other means, thus checking the calculation. This essentially is a characteristic of UNICORN, rather than Telex systems in general.

Submission of a Completed Forecast

Selection of the SUBMIT option from the main "menu" causes a message to be sent to the UNICORN steward, advising him of the fact that this user has run SUBMIT. This is achieved through a standard Wellcome internal mailbox system.

Two vital safeguards are built into SUBMIT. The first of these is a method of ensuring that the user has not input changes to the data since last running COMPUTE, and is achieved by flag setting and unsetting in the two routines respectively. If the flag is found to be set, SUBMIT generates an appropriate error message to the user, and refuses the submission. The second safeguard is to ensure that the key IDs have been listed by the user, so that Corporate Planning can be certain that the user is aware of the results of the calculations upon his data. This is the purpose of the CHECK option in PRINT; again a system of flag-setting is used so that the submission is rejected with an appropriate error message if CHECK has not been selected within PRINT since the last COMPUTE.

A further possible action which was considered for SUBMIT was the automatic shutting-off of file access to the user, to prevent subsequent alteration. However, the frequency of last-minute revisions experienced in practice meant this idea was rejected.

Telex users are not required to forward a hard copy confirmation of their data, the invocation of SUBMIT is all that is required. A number of users have requested that Corporate Planning return to them a set of facsimile reports after receipt of the data, which of course presents no difficulty.

Logging Off

The OFF option, as described earlier, logs the user off by calling WALKAWAY and returning with naked right-arrow to immediate execution. An appropriate latent expression is set for the CONTINUE workspace, thus making the system self-perpetuating, and a mechanism of "System Version Numbers" ensures that Telex users always get the most recent version of the UNICORN "pseudo-workspace" when they next log on, even though it may be 3 months or a year later]Mathieson Shaw and White 1980[.

Additional System Developments

One of the potentially most baffling events which a Telex user might experience would be a "break-in", causing suspension of execution. Guarding against this was unfortunately impossible in the first release of Telex UNICORN, but became possible with the introduction of Event Trapping. A Telex user can generate an interrupt by striking any key or keys rapidly while the Telex machine is printing. The action taken on a user interrupt is simply to return to the main "menu"; since it is very unlikely that an interrupt will occur other than by a user deliberately aborting during printing, this action is generally appropriate.

UNICORN provides a standard HELP facility; if a terminal user types HELP at any stage, whether in response to a prompt or not, he receives a brief one-line help message, from which he may if he wishes then go on to request a fuller explanation. It was obviously essential to ensure that if a Telex user invoked HELP he would receive clear, concise instructions telling him exactly where he was and what he should do next. Accordingly, the action of the HELP function had to be modified for the case of a Telex user. In addition, as data input from paper tape is performed by direct quote-quad mode, the INPUT routine must scan each line as it is received for the word HELP before storing it in the internal buffer, printing an appropriate message if necessary.

As a check on the difficulties experienced by Telex users, the HELP routine also, unknown to them, sends a message via our internal mail system to the UNICORN stewards, notifying them of the error message returned to the user and the state of his)*SI* stack.

Further ways in which the progress of users is monitored by the UNICORN stewards are, firstly, the automatic generation of internal mail messages whenever a Telex user logs on or off (the latter only when he does so voluntarily via the OFF option, line drops not being trapped), and secondly a copy of any serious error messages he receives from the system. In this way a user in trouble can quickly be identified, and corrective action taken.

A problem which can occur in systems of this kind is that of language. In the case of UNICORN, this did not arise, as English is the "working language" of Wellcome, and one can be confident that users will be proficient in it. In the general case, however, thought may need to be given to the provision of message files in other languages.

User Documentation & Training

A user manual had to be written specifically for Telex users, as although very comprehensive documentation already existed for terminal users, it was felt that a simplified guide would be easier to follow. The Telex User's Manual contains approximately 30 pages, with sections on every possible stage of using the system, from the completion of the forms to Error Recovery.

Experience has shown that user training takes approximately 2 days, and to be effective must be performed at the user's location. Although not always the case, the Telex is usually operated by a senior secretary, generally that of the General Manager or Finance Manager of the company in question. The responsibility for the preparation of the figures would generally lie with the Finance Manager. This means that two people need to be trained in different aspects of the system, with the additional complication of needing to identify the point at which the advice of the other should be sought. This is in marked contrast to the UNICORN terminal users, who are all of them senior financial executives operating the system themselves. It is probable that this is a result of the traditional function of the Telex machine in the office environment, but whatever the cause, it is a fact of life and must be accommodated.

Other Systems Using Telex

Once the UNICORN system had been successfully adapted for Telex use as described above, the Corporate Planning Directorate became involved with a related application. This is a system for collecting preliminary estimates of key performance figures from all overseas subsidiaries immediately following the end of each quarter, enabling the Board of Directors to obtain a rough guide to Group results some 4 weeks or so before true "Actuals" become available.

Technically this system, known as the Flash System, is extremely simple, as it consists of merely collecting approximately 20 numbers and a check total from each unit, with no calculation or validity checks. The benefit of this simplicity is found in the fact that it was possible to bring the system into operation merely by sending to all units a two page set of instructions, with no training visit; the success rate on the first attempt was well over 90 percent, with the remaining problems virtually all being concerned with availability of Telex machines or lines.

One aspect of this system in which it differs significantly from the Telex version of UNICORN is that all input is expected to be typed from the Telex keyboard, and not input from paper tape. This is in order to be able to include users whose machines have no tape facility, as well as keeping the operation of the system by the user extremely simple.

Again, once the data is in the machine, it is examined, additions are made in respect of non-local items, and reports are produced. Currency conversion and consolidation are carried out, and Group level reports are produced and presented for Board review within about a week of submission. Even a week might seem in some ways excessive here, but it should be explained that the principal cause of delay is the need to add non-local data dependent on the submission, and often considerable time has to be spent in consultation for this.

Although technically somewhat dull when compared with UNICORN, the implications of the Flash system are in some ways vastly more exciting. The successful operation of the system is evidence that it is possible to collect data on-line from anywhere in the world where there is a Telex machine, provided that the systems used do not make excessive demands on the users. The data can, in a properly designed system, be rapidly processed and evaluated, giving the potential for much swifter action and more effective decision-making by corporate management. At the same time, in the case of systems which provide some kind of feedback, another tool is provided for local management, so that benefits can be realised throughout an organisation.

An Assessment of the Success of the Telex Systems

Overall there can be no doubt that the Telex systems implemented by the Corporate Planning Directorate have proved highly successful. Operational problems have been very few, being confined generally to shortage of Telex lines and occasional line failures in some of the more remote parts of the world. Problems of line noise have, somewhat to our surprise, turned out to be virtually nil. The user reaction has been generally favourable, and in some instances very enthusiastic; this is no doubt partly due to the fact that Corporate Planning share the time saved with the Reporting Units, relieving them of a little of the end-of-period pressure they inevitably experience. This time-saving has worked out at about 2 weeks, of which perhaps 2 days are given to the Reporting Units; the saving is as much as was aimed for, and is of real value in enabling earlier Board review, and if necessary action than was formerly possible.

The accountants within the Corporate Planning Directorate have found that the technical quality of the plans and forecasts has been maintained and in some cases improved with the introduction of Telex access. Occasionally, however, there are still problems where items arise whose treatment is not adequately covered by the standard forms. Such problems usually need resolution by consultation with the Reporting Unit; the reason why they are perceived as problems is that the Telex systems are designed with a strong bias towards simplicity, so that although UNICORN can handle such exceptional problems as, for example, converting a Branch Office to a Subsidiary Company, the method of doing so is not covered by the standard documentation. In practice, such problems are very rare, but when they do occur, the course followed is for the Reporting Unit to submit the basic data, and any esoteric adjustments are made afterwards by Corporate Planning.

The costs of using Telex access have also turned out to be approximately as expected. Currently 10 Reporting Units submit forecasts regularly by this method; an analysis of the total connect times for these units in the most recent exercise shows a mean of 40 minutes with a standard deviation of 12.5 minutes. The mean number of total lines of local data in each of these forecasts was 272, of which 110 were basic input lines and the rest were derived. Each line contained 8 data items. For the most recent Flash exercise, in which 30 Reporting Units submit via Telex, the mean connect time was 8 minutes with a standard deviation of 5.25 minutes. The connect times for the Flash are perhaps a little higher than one might expect, but this is probably an inevitable consequence of the decision not to use paper tape input for this system.

Conclusion

We have shown that the use of Telex direct access for the transmission of reasonable quantities of data and manipulation of that data by interactive computing is a practical possibility. Careful attention to systems design, especially in the areas of reliability and simplicity, is even more important than usual, and the cost-effectiveness of the system as well as the desirability of the re-distribution of workload can only be determined by the management of an organisation proposing to use this approach. Used sensibly, however, such systems undoubtedly have the potential for improving monitoring and control of widely dispersed business operations, and from discussions with I.P. Sharp as well as other computer users it is clear that this method of access, which effectively gives a network across the entire wold, is certain to grow.

Reference

D.G.I. Mathieson, R.W. Shaw and J. White, "Civilising APL: An Approach To Integrated APL Systems", **APL-80 Conference Proceedings**, June 1980.

MANAGING APL RESOURCES AT UPJOHN

Richard J. Busman
The Upjohn Company
Kalamazoo, Michigan

Our success in managing APL resources is determined by the quality of service to the customer in on-line response, development of new systems, and maintenance of existing systems. Upjohn has discovered that the path to quality customer service and successful resource management has many obstacles.

This paper is an overview of what Upjohn is doing to manage APL resources and provide quality service to customer units outside of the computer services division. It should not necessarily be construed as the philosophy of how it should be done.

APL has been the forerunner in the introduction of the on-line environment for development and production processing at Upjohn. Changes in computing philosophy have occurred to include on-line in the mainstream of business systems implementation. The planning and managing of APL resources is difficult because of short implementation timeframes for business systems. Shortened time frames make it more difficult to plan hardware needs, obtain and train staff, and deal with other areas in the organization.

The challenge is to provide adequate customer service within the constraints on resources such as computer hardware and personnel. (This limited computer hardware constraint will be less of a problem in the future due to dramatic decreases in hardware cost.) Challenges in other areas are to prioritize the development of systems by ranking them according to the higher corporate payback and to develop a business system environment for APL with access to all corporate data using hybrid systems where appropriate. (At Upjohn we have been emphasizing the business environment and have not placed enough emphasis on the time sharing aspects, such as on-line response times and system uptime.)

The APL resources being managed at Upjohn fall into the following areas:

- APL System Development and Maintenance
- Operation of APL Systems
- APL Hardware/Software Environment

To discuss the overall management of APL resources we need to look at the organization supporting APL customer service and see how APL is managed in each area. This supporting organization at Upjohn is called Information Systems and Computer Services (IS&CS) and consists of groups A to L as outlined in Figure 1. (Not shown are customer areas such as Auditing, Chemical, Industrial Engineering, and Marketing Systems who are using APL for developing personal computing systems.) This organizational structure helps us provide the best APL service. In the period 1975 to 1980, APL has become an integral part of our business systems environment and the organization has evolved to allow that to happen. We develop imaginative on-line systems using the latest tools available, such as laser printing, microfiche, plotting, CRT displays, etc. We have an organization and environment that provides complete data processing facilities for APL.

As you can see in Figure 1, many of the units in our organization have some portion of the responsibility for providing APL service. APL analysts interface with several areas to provide the support essential to customer service, and have the responsibility to be able to deal with people as well as problems.

The life cycle of an APL project developed by IS&CS for a customer area likely would be:

- customer request via a Divisional Services Manager (A) or as the result of preliminary problem solving by the Management Science or Mathematical Services units (C)

- referral to APL Design and Development Group (B)

- assignment of account numbers and billing by Corporate Time Sharing Administration (E)

- system resource impact to software support and operations (B,C)

- development of project with close customer interaction (B)

- documentation of system for turnover to APL Support (D)

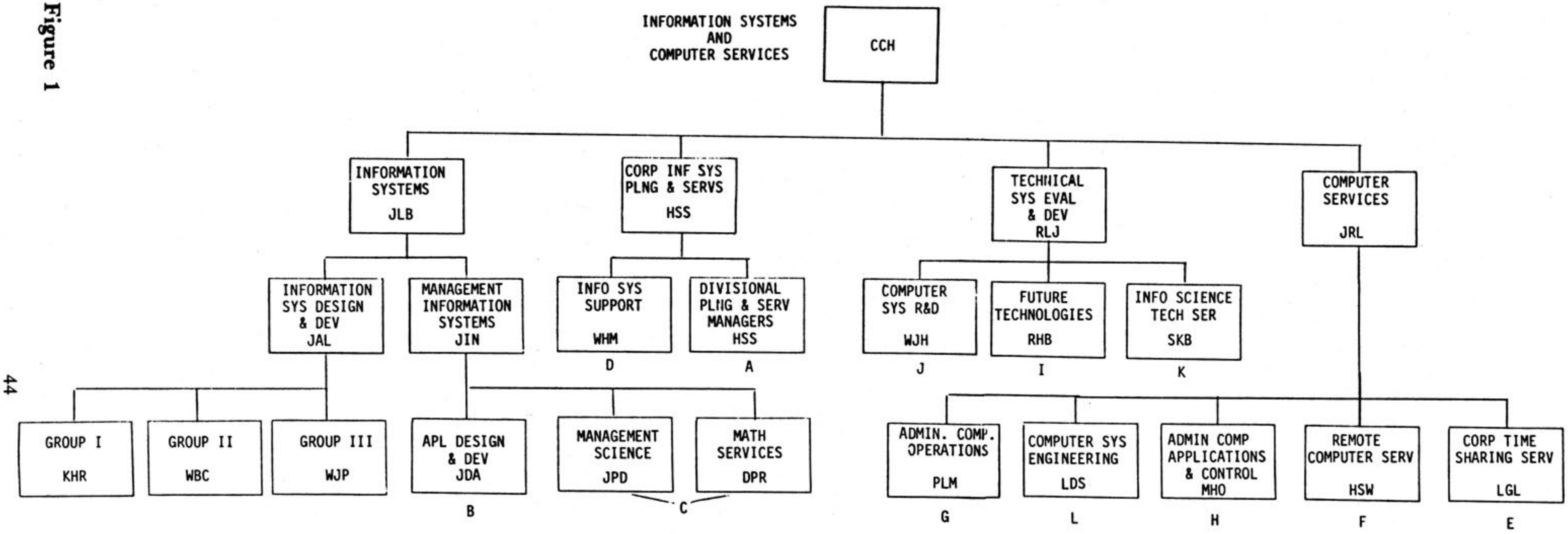
- documentation and turnover for nightly APL production (H)

- monitoring of system impact by the Communications Control Center (F)

In addition to the above interfaces, the APL analyst consults with the software and the communications staffs in solving problems that occur during development. Also, the analyst may be appointed to be a member of various task forces to guide the short or long range APL direction in software and hardware areas.

The following sections will cover each area of IS&CS in relation to its responsibility in managing APL resources.

Figure 1


INFORMATION SYSTEMS AND COMPUTER SERVICES
CCH
INFORMATION SYSTEMS JLB
CORP INF SYS PLNG & SERVS HSS
TECHNICAL SYS EVAL & DEV RLJ
COMPUTER SERVICES JRL
INFORMATION SYS DESIGN & DEV JAL
MANAGEMENT INFORMATION SYSTEMS JIN
INFO SYS SUPPORT WHM
D
DIVISIONAL PLNG & SERV MANAGERS HSS
A
COMPUTER SYS R&D WJH
J
FUTURE TECHNOLOGIES RHB
I
INFO SCIENCE TECH SER SKB
K
GROUP I KHR
GROUP II WBC
GROUP III WJP
APL DESIGN & DEV JDA
B
MANAGEMENT SCIENCE JPD
MATH SERVICES DPR
C
ADMIN. COMP. OPERATIONS PLM
G
COMPUTER SYS ENGINEERING LDS
L
ADMIN COMP APPLICATIONS & CONTROL MHO
H
REMOTE COMPUTER SERV HSW
F
CORP TIME SHARING SERV LGL
E

A - Planning for current/future project development/support
B - APL Design and Development
C - Extensive use of APL by Management Science and Math Services
D - APL Application Support
E - Corporate Time Sharing Administrator
F - Communications control Center/software problem monitoring
G - Operations
H - Nightly APL production
I - Future Hardware planning
J - Software support
K - Future data base interfaces
L - Evaluation of small APL minicomputers

APL System Development and Maintenance

The next four areas deal with the development and maintenance (support) of applications as managed by the following units:

A - Divisional Project Planning and Services Managers
B - APL Design and Development
C - Management Science and Mathematical Services
D - Information Systems Support

A) Divisional Project Planning and Services Managers

Introduction

Each divisional services manager is responsible for coordinating the information needs of one or more divisions with the appropriate IS&CS resource unit. This includes both short range planning (annual budget) and long range planning (three year horizon) of systems, coordination and monitoring of system development activities for a division, and coordination of maintenance and operation activities for existing systems. Their success is judged by customer satisfaction. Since the customer pays for the system via a direct charge system to be described later, the customer wants to minimize development and operating costs and wants it developed as soon as possible. The services manager must weigh the trade-offs in development cost against operating and support costs.

The services manager desires to provide the following environment for the customer:

Business System Environment

This environment consists of the use of video and hardcopy terminals, on-line or deferred/detached processing, high quality and highspeed printing (such as IBM's 3800 printer), and the sharing of data between technologies such as APL and data base management systems like IBM's Information Management System (IMS). The challenge is to find hardware and software to meet these needs. (Each installation likely will need to write auxiliary processors to satisfy particular needs.)

At Upjohn, APL systems have become some of the largest users of CPU resources. For example, we have a Cost Accounting System that costs $330,000 per year in operating costs and a Production Planning System that is $200,000 per year. These systems have evolved due to increasing requirements, from systems costing approximately $25,000 per year. Their size requires that they be carefully managed. A large percentage of the cost is in CPU time to manage and access data. (We need new data accessing tools to bring these systems to more manageable CPU usage levels. In the Production Planning System in particular I believe that we could reduce operating costs by examining the code and rewriting it where appropriate for greater efficiency in file accessing and processing data. In general, systems that have evolved in development should be examined for improved efficiency.)

The use of APL in the business system environment has grown because the customer pays for information system development work via the following direct charge system:

Direct Charge System

All work by IS&CS for any division is billed on an analyst-day rate. Thus, customers are held accountable for their expenses for computer systems. Very often customers will request that two estimates be provided; COBOL and APL. Each Fall, customers budget for development projects, maintenance, and operating costs in a process called "Work Order Credit Budgeting". These budgets are used by IS&CS to plan the number of analysts required to staff each unit. Cost accountability and more rapid implementation cycles have led to an increasing acceptance of APL as a cost effective development approach for both small and large business data processing systems.

B) APL Design and Development

Introduction

The APL Design and Development unit provides most of the APL development work as a centralized technical group. More decentralized development will occur as we make utilities such as those for storing and reporting departmental data and corporate data more readily available from APL. As analysts switch jobs and move to customer areas, we will also see an increase.

Currently, we have 10 APL analysts and 2 contract programmers. The APL unit was formed to provide better support of APL systems implemented and being developed. In contrast we have 33 people in the COBOL development group; in the support groups, we have 1 APL and 25 COBOL support analysts. Upjohn spends $2.4 million in analyst time for developing COBOL systems; $500,000 for APL. The APL unit was initially staffed with members of our Management Science and Mathematical Services units which also number 12 analysts.

The APL Design and Development Unit is managed by a head who is assisted by three group leaders who help manage projects in specific areas of the company. Once a month, forecasts of project activity by analyst are prepared by the three group leaders. Also, highlights are submitted monthly, to the head, by each analyst. Each week analysts input time spent on projects into a project control system for keeping track of time spent and for billing work to customers.

In order to develop quality systems, we need talented people. The skills and training required are described below:

APL Analysts

Most of our analysts have a mathematical aptitude with degrees in math, statistics, or related fields. Course work in matrix algebra and computer science is helpful. Since the analyst works closely with customers, other important characteristics are responsibility, creativity, enthusiasm, and oral and written communication skills. Some of the analysts with a background in COBOL/ FORTRAN systems development, have been trained in APL, and successfully developed complex APL systems. Even though smaller project teams are used to develop systems, skills in working with other people are very important because of the many interfaces with the customers and IS&CS organization. In addition to solving problems, the APL analyst must be an effective project leader when dealing with people.

The people we hire generally have no prior experience in APL; it is difficult to find experienced APL people who want to move to Kalamazoo. So we train them using a one-on-one tutoring approach with a tutor selected from the APL unit. We use three IBM instructional courses in combination with the book **Introduction to APL** by Gilman & Rose. The student is given wide latitude in the approach. For example, some may use IBM's on-line APL course exclusively; others may use the book almost exclusively. At the end of the course a final exercise is required to give the student practice in the use of files and public library routines. Longer range training is accomplished by the group leader working with individual analysts, through workshops conducted by the staff, and other information science training programs. The instruction is coordinated by our IS&CS training coordinator (not shown on IS&CS chart).

Our success in developing systems speaks well for APL as discussed below:

Application Systems

We have been successful at developing major projects using a beginning APL'er directed by a senior staff member. Usually a project proposal is prepared by a senior staff member detailing the solution, major design specifications, cost estimate, and implementation plan.

Detailed requirements such as report formats, input edits, etc., are worked out with the primary customer during the development phase. This evolutionary development approach is a key to resource management and our successful implementation of APL systems. The standard approach with COBOL systems using accepted project management techniques is to ask the customer to indicate all requirements and specifications prior to the development phase. Any changes occurring during development require a formalized change procedure to gain approval for change and, of course, adjustment of project cost estimates. Customers like the evolutionary approach to system development.

In June 1975, APLSV was installed on our administrative center test computer so that we could develop in-house systems and migrate work from external time sharing. Since then, the business and support groups at Upjohn that maintain systems using APL include Accounting and Finance, Administration, Agricultural, Auditing, Control, Fine Chemicals, Healthcare, International, Lab Procedures, Pharmaceutical Marketing, Personnel, and Pharmaceutical Production. Areas that could benefit from use of APL but are not using it are Pharmaceutical Medical Affairs and Pharmaceutical Research and Development. The latter area operates an independent computer center and the statistical groups there have developed extensive FORTRAN libraries; in addition, special data base systems storing clinical research data have been created. Therefore, APL has not been particularly attractive with the Pharmaceutical Research and Development area because of the difficulty of interfacing APL with FORTRAN statistical libraries and special data bases.

The magnitude of the APL systems implemented range from small personal computing systems to a large Production Planning System supporting nine terminals accessing and jointly updating the files, costing $200,000 per year in operational cost. In April, we implemented a Sales Analysis Reporting System which prints 20,000 pages of reports per month. This latter project was developed as a hybrid system, with COBOL used for creating summarized data at the sales area, district, and territory level by product and product classification. APL is used to create summarized files, select report formats, and produce reports. The hybrid approach was chosen because:

a) the customer previously used APL very successfully in a Drug Distribution Data Reporting System, so knew of APL capabilities;

b) we needed to develop the system quickly due to installation of the new computer not supporting Autocoder in which the old system was written;

c) the volume of data from the order entry system could best be handled by COBOL;

d) the 240 separate report formats could best be done by using the power and interactive nature of APL.

APL is being used to develop large, traditionally COBOL business systems, including reporting systems (Sales Analysis) and recordkeeping and planning systems (Production Planning, Fine Chemicals Materials Systems, and Cost Accounting).

For some statistics on in-house APL usage:

Number of enrolled users: 185
Number of video terminals: 41
Number of hardcopy terminals: 50+
Daily connect time (as of April 1980): 117 hours
Yearly Billing (last 12 months through April 1980): $524,000

See Figure 2 for a graph of APL CPU Seconds by month, from January 1976 to April 1980. APL has grown by a factor of 15 since January 1976. For the last four years, APL CPU usage has increased at a 71% compound growth rate.

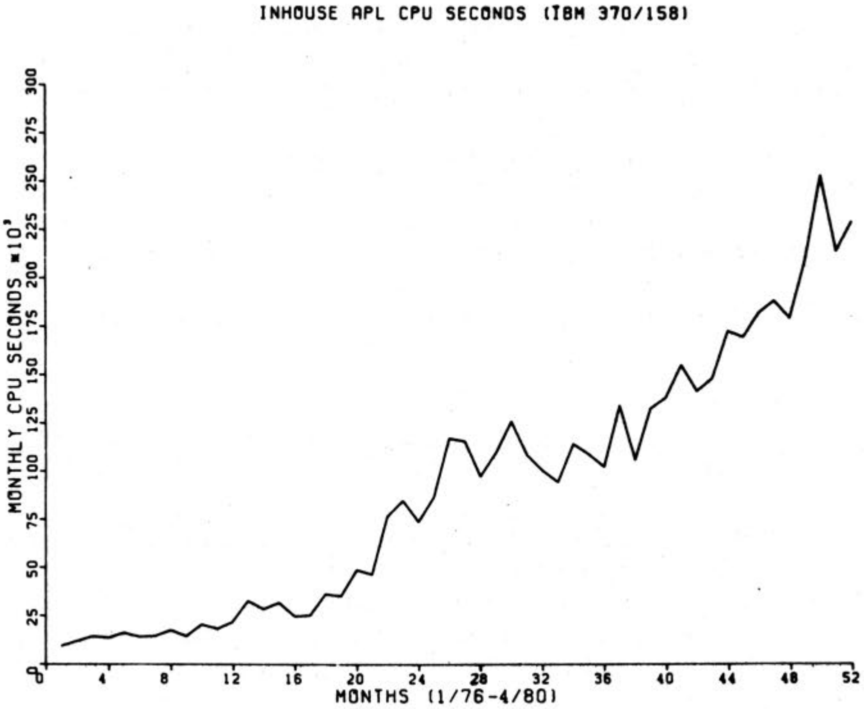


Figure 2

In order to increase productivity and maintain consistency between applications we've developed extensive public libraries as described below:

Public Libraries

The availability of a variety of efficient and well-documented utilities stored in public libraries increases the productivity of the staff.

For example, we have utilities for:

- accessing AFM (emulates APL*PLUS file subsystem) and TSIO (standard IBM) files
- writing conversational programs
- cross-reference listing for workspace documentation
- data editing and processing
- editing functions in a workspace
- fullscreen editor for video terminals
- plotting
- sorting data in batch
- text editing
- report formatting functions
- printing reports on a highspeed printer

A fullscreen editor is used in making changes and additions to current public library documentation; revised documentation can be printed on a 3800 printer on request. We are planning on setting up a documentation retrieval system by keyword for making public library documentation more readily available.

To keep analysts abreast of new developments in APL software and utilities, technical workshops are held on a monthly schedule so that techniques and utilities can be communicated and documented.

In order to maintain systems, we've developed a standard for documentation as described below:

Documentation Standard

Formal documentation is divided into the following four parts:

- system
- user
- operations (if nightly processing by application coordinator)
- technical

Procedures are outlined for turnover to the operations and support staffs.

A support protocol is followed for all problems and changes to turned-over APL systems. This involves the submission of a support request form to the divisional services manager detailing problems or changes required. The services manager forwards it to the support staff. The support staff assigns a project code and task number for tracking the work and charging it back to the customer.

Documentation for each project must be completed before the project is officially done. A project completion report detailing cost versus budget signals project completion.

C) Management Science and Mathematical Services

Both units provide mathematical, statistical, and operations research consulting on a corporate-wide basis. The Management Science unit serves business areas while the Mathematical Services unit supports production areas.

These units use APL extensively along with FORTRAN, SAS, and SPSS. Projects initiated by these units may grow in scope and be implemented by the APL Design and Development unit.

D) Information Systems Support

All systems documented according to standards, can be turned over to this unit. Thus, responsibility shifts from the development unit to the support unit. Any changes or problem fixes are handled via a support protocol as follows:

- problem or change detailed on a support request form and given to divisional services manager

- services manager may obtain cost estimate and approves work

- accounting number assigned for progress tracking and customer billing

- task assigned to APL support person for implementation and documentation update

Thus, changes to operational systems are carefully managed and controlled. To maintain efficiency, maintenance changes and enhancements involving less than one analyst's day of effort do not require the above record keeping procedure. Staffing the support unit with trained APL professionals is proving to be a significant problem.

Operation of APL Systems

The section that we just finished dealt with the development and support of applications. Next we turn to the operation of APL systems as managed by the following units:

E - Corporate Time Sharing Services
F - Remote Computer Services
G - Administrative Computer Operation
H - Administrative Computer Applications and Control

E) **Corporate Time Sharing Services**

The time sharing administrators (one for internal and another for external) enroll new APL users and set up the account environment including the amount of file space requested. The external administrator keeps track of the account numbers on the system and a system written in APL is used to bill customers each month for computer and terminal costs. The system prepares bills by unit with YTD information and management summary reports by project. This system allows the comparison of projects on a cost per connect hour basis.

In addition, the CTS administrator purchases and maintains dialup terminals for customers including paper and ribbon supplies.

F) **Remote Computer Services**

A communications control center has been established to monitor on-line performance and serves as the focal point for user problems via a "hotline". APL response is monitored on a daily basis and a report is issued each month to upper management comparing observed response against pre-set goals. The Communications Control Center (CCC) handles analyst or customer inquiries as to why response is slow or non-existent. The CCC operator handles file space allocation increase requests. The CCC operator will bring the system up in the morning and take it down in the evening to secure the necessary file backups.

A communication team meets weekly to solve problems that affect users such as software bugs, communication line and terminal problems.

Plans are underway to provide a communications network so that the same video terminal can be used to access APL, IMS, and TSO. Currently we can access APL and TSO from the same terminal. We are running ACF/VTAM with plans for an update to cross domain ACF/VTAM in 1981.

Planning for additional APL compatible CRT/printer devices and the related hardware (IBM 3705, etc.) is also a key responsibility of this group.

G) **Administrative Computer Operations**

The Administrative Computer Operations unit is responsible for daily service as provided by computer operators. The computer operators attempt to minimize on-line interruption due to other technologies (batch, TSO, IMS test). In the evening, an operations operator executes any APL jobs that were scheduled that day by an application coordinator.

H) **Administrative Computer Applications and Control**

An application coordinator supervises the execution of those jobs which the customer does not want to run or cannot run because of:

- long elapsed time of run (elapsed times up to 8 hours on the Production Planning System, for example)
- desire for second shift discount (25% discount)
- need to schedule with other non-APL processing
- need to better manage and schedule use of resources by large applications
- need to use report distribution facilities, report checking, microfiche, plotter, etc.

Any off-loading of applications to non-prime time will improve prime time on-line response times. Functions that require little user interaction and do not need quick turnaround are good candidates for deferred processing.

If a system is documented according to the operations documentation checklist, it can be turned over for scheduled nightly production.

Since no deferred processing capability for APL yet exists at Upjohn, the systems are executed by a computer operator using a manual procedure. Plans are underway to develop such a capability which will reduce the need for the application coordinator function, but will not eliminate it.

APL Hardware/Software Environment

Now that we have completed a review of the operation of APL systems, we'll turn to the last section dealing with the system environment for APL as managed by the following units:

I - Future Technologies
J - Computer Systems Research and Development
K - Information Science - Technical Services
L - Computer Systems Engineering

I) **Future Technologies**

The Future Technologies unit plans for future APL hardware needs by using the following inputs:

- monthly updates supplied by the APL Design and Development unit on projects under development

- yearly APL time budgeted by customer units according to the fourth quarter work order credit budgeting report

- monthly report to upper management on percentage of CPU usage by technology (batch, APL, IMS, TSO) for current month and last 8 months

- monthly APL response monitoring report

In August, hardware plans are fixed for the next year based on linear projection of utilization with room for unprojected growth.

We plan for CPU capacity, local and remote ports, and disk capacity, and we create a cost center for APL based on hardware, software, and support costs. Then we establish rates (CPU, Connect, Execute Channel Program (EXCP)) based on estimated usage so that the costs will be recovered without a profit. The CPU and EXCP rates are the same for all technologies. Any extra cost is absorbed in connect time rate. With increased use of CRT's, a goal for 1981 is to eliminate charging for connect time.

At an Upjohn Time Sharing Conference held in March 1980, the Director of the Information Systems and Computer Services (IS&CS) Division, referred to our in-house time sharing as "illhouse" time sharing. The word "illhouse" appeared in a rough draft of his speech, as he described it, either as a Freudian slip or typographical error. Even though our workload projections over the past three years have been fairly accurate, as of March 1980, our equipment was at capacity. The problem was that we had coordinated the installation of new hardware with the move to a new building housing computer personnel and equipment that was originally scheduled for early 1980. Unfortunately, the construction was delayed several months. Because the new hardware is water cooled, a decision was made not to invest the money required to install the hardware at the current site. Thus we had to endure an extended period of "illhouse" time sharing.

J) Computer Systems Research and Development

Introduction

This software support unit is responsible for the APL operating system software. One person on the staff is primarily responsible for keeping APL up-to-date on vendor changes and also solving problems that occur in day-to-day operation. A formal problem-reporting protocol is used for problems that analysts or customers uncover. These problems are then assigned to the Communications Problems report for followup.

All system tuning and management of disk space is handled here. This unit has been the major force in the direction of APL technologies as discussed below:

Decision to Convert to VSPC from APLSV

Upjohn is currently running VSAPL under VSPC. TSO, IMS on-line testing, and batch also run on the same machine as APL. Previously, APLSV was used, but in order to satisfy the following two aims:

- one technology doesn't impact another, and
- one user within APL doesn't adversely impact other APL users,

the decision was made to convert to VSPC. (VSPC didn't completely solve the above two problems because of the paging that is added to the system. Also, we have had instances where one user can adversely affect others.)

Other reasons for converting to VSPC were to move to the mainstream of IBM development (Class A product) and to use one communications network for APL and TSO.

Conversion to VSPC/VSAPL

The conversion from APLSV to VSPC went quite smoothly from an APL standpoint, but problems with VTAM and the support of dialup terminals caused severe disruptions. (The support of dialup that existed in the latest version of VTAM was not in our version of ACF/VTAM.) We spent 4 to 5 months before we solved the major problems. We currently have a problem in setting tabs on hardcopy terminals. Also, our attached printer support for video terminals is not as good as APLSV's.

In workspace conversion we had problems in:

- conversion of AFM files (Boolean's in access matrix)

- format (e.g., 3 2 ⍕ 0 gave *DOMAIN ERROR*)

- a function used to convert to numeric from packed decimal

- *A*←⍞ (no beep signals ready for input)

- 5 or more 9's (i.e., 99999) input on dialup signed user off system

- APL system errors

- shared variable timing problems in AFM and TSIO

- functions where a label and the name of function were the same

After several months of plannning, eight analysts converted approximately 1000 workspaces (170 public library; 300 using AFM) from APLSV to VSPC on a single weekend including the conversion to the new single shared variable version of AFM. (The previous version used a different shared variable for each file tied.) Each analyst was responsible for a group of workspaces. Conversion was particularly easy due to the use of programmed PF keys on the video terminals which were programmed by user written APL functions that examined each workspace to be converted and programmed the PF keys accordingly.

The handling of the APL CONTINUE workspace (WS) under VSPC was rather peculiar. If a CONTINUE existed at sign-on, it would always be loaded as the active WS. At sign-off, it would always be saved. We modified the system so that a pre-processor warns the user of the existence of a CONTINUE WS prior to actual sign-on so a decision can be made to stop before signing on, thus saving the CONTINUE. Also, the CONTINUE WS is automatically deleted at sign off which is consistent with the handling of non-APL CONTINUE WS's.

Future Direction

A task force has been established to determine the future direction of time sharing at Upjohn in relation to current work in-house and on outside time sharing sources.

Factors to be considered are:

- standalone or mixed environment
- communications between technologies
- ease in conversion of both in-house and external systems

K) Information Science-Technical Services

Current research is under way by this unit to investigate the use of IMS data bases by APL. All current applications either use VS TSIO which provides sequential or direct access of standard OS files or AFM which is an auxiliary processor that emulates the APL*PLUS file subsystem. Any interface now with IMS involves using another technology to extract the data from IMS and create a VS TSIO file, or vice-versa. In some cases, we use COBOL or PL/1 programs to directly update APL objects stored in VS TSIO files.

The area of data base access is definitely one that could be of great benefit in reducing operating costs and improving service levels for large APL applications. Particular problems that we have in larger applications are:

- managing large directories of several thousand items which causes extra file accessing to find the location of an item;

- the simultaneous updating of multiple files by multiple users which causes poor on-line response time due to file holding. (Holding at the component level would help);

- the accessing of scattered data in multiple files from several systems, causing many files to be tied and accessed.

Also, an increase in productivity could be realized because it would relieve the APL programmer of the burden of managing pointers and directories. The availability of a keyed access method may also help. (We need to investigate whether VSAM would provide a viable keyed access method.)

In addition to data management, the Information Science — Technical Services provides utilities to improve APL productivity.

L) **Computer Systems Engineering**

This unit is responsible for the use of minicomputers in automating production processes. Its contact so far with APL was in the evaluation of APL minicomputers for a customer in marketing. If APL becomes more readily available for minicomputers, this activity will undoubtedly increase.

Conclusion

Success in managing our APL resources is judged by the quality of service to the customer in on-line response, development of new systems, and maintenance of existing systems.

An overview of the management of APL resources at Upjohn was described by looking at the organization that supports APL services and discussing the responsibility for APL service that each unit has and the interfaces that the APL analyst has with many different areas. The organization helps solve problems in providing APL service. In the period 1975 to 1980, APL has become an integral part of the business systems environment at Upjohn. The organization has evolved to support the management of APL resources and that evolution will continue in the future.

Many factors can affect the quality of APL service including even some outside factors such as moving the computer center to a new building.

Cost accountability and more rapid implementation cycles have led to an increasing acceptance of APL as the more cost effective development approach.

APL TECHNOLOGY: LANGUAGE, ARCHITECTURE, SYSTEM CONCEPTS, AND APPLICATIONS METHODOLOGY

Adin D. Falkoff
IBM Research Center
Yorktown Heights, New York

Introduction

APL is more directly rooted in mathematics and algebraic notation than are other general purpose programming languages, and has developed with a greater degree of independence with respect to machines (Ref. 1,2). Its qualities as a tool, as well, are sufficiently different to have engendered a radically different mode of operation in professional programming establishments that use it extensively (Ref. 3). This suggests that it may be worthwhile to study the model of information processing implied by APL: to view APL as a **technology** — a systematic approach to information processing and a logical framework for structuring the field — and see where this may lead.

The principal elements of the APL technology are: **arrays** as primitive data objects, **function** syntax and semantics, **operators** for producing derived functions, exclusive use of **names** as identifiers, communication via **shared variables**, and **workspaces** for mediating the computing environment. Each of these holds implications for the architecture of hardware processors, the design of data processing systems, application design and methodology, and language design itself.

This paper is organized with a section on each element of the APL technology, discussing its significance for those aspects of information processing to which it appears to have the most relevance.

Arrays

The primitive data objects of APL are arrays. While all programming languages deal with arrays somehow, most of them recognize only single elements and do not consistently recognize the arrays themselves as entities. That APL does, is shown most simply by the presence of the shape function, which returns the size of its array argument as the result. A most important property of an APL array is that it is purely abstract and has no personality. A vector of ones and zeros, for example, is not automatically taken to be a binary number, although the number it may represent can be easily produced by means of the decode function; and a matrix, though it may contain the necessary information and be used to represent a relation in a data base, is not of itself a relation.

Arrays are pervasive in all aspects of data processing, but their implications for machine architecture are of particular interest; it is at least suggestive, in an APL context, that large scale integration must necessarily work in terms of arrays at the innermost hardware level. For example, it is likely that with APL in mind logic could be designed to take better advantage of array operations so as to limit the traffic on and off a chip. On a higher level, an obvious possibility is to build arithmetic and logic units that accept arrays as operands and return arrays as results. This is not a new idea; array processors as auxiliary devices are well known, and a number of machines have had integral special purpose array processors, ranging from pipeline design to arrays of scalar arithmetic units operating in parallel. But none of these was designed for an APL environment, wherein control information is automatically available and all the primitive functions are inherently array operations.

Application design in APL is, of course, strongly influenced by arrays, and their presence is unquestionably an important factor in the productivity of APL. Arrays are central to the expressiveness and conciseness of the language, and programs conceived as transformations of arrays lend themselves more readily to analysis, and deductions about correctness, than do equivalent programs written in terms of single elements.

Thinking about problems in terms of arrays often provides insights that are not otherwise available. Two complementary points are worth noting: first, it is a mistaken idea — based on experience with other programming languages, perhaps — that thinking in terms of arrays is less "natural" than thinking in terms of scalars. Contrary examples abound: "pick the ripe cherries", "plug the holes in the dike", not "examine the cherries one-by-one and pick each one if it is ripe", etc. Second, even among those who habitually think in terms of arrays, the benefit of using arrays with very few — particularly, two — elements along a given axis, is sometimes overlooked. Used with the relational functions, for example, such arrays often reveal symmetries that radically simplify the logic of a problem.

Functions

Primitive functions in APL are defined independently of the representation of their arguments, have no hidden side effects, and have explicit arguments and results. This last is essential to the expressiveness of the language, since it makes possible the formation of compound statements. Furthermore, defined functions are syntactically like primitives, and when designed with explicit arguments and results can be used freely in combination with them. Meaningful statements of reasonable length tend to be the norm under these circumstances, which, in terms of machine design, leads to the possibility of a level of parallelism above that of the array. A machine to take advantage of this might have statement decoding overlapped with other operations, as well as look-ahead within a statement to afford substantial parallel operation without some of the difficulties associated with look-ahead in scalar-based machines.

The functional approach has had a profound effect on the design methods used in application programming, and it is certainly one of the principal factors in achieving the high productivity rates experienced. Application systems developed with this kind of facility tend to be composed of independently executable and replaceable units, which are combined to form subassemblies and assemblies in the same way that complex mechanical systems are constructed.

Because individual units have explicit input and output they can be tested and verified independently, before being combined to form more complex units, and the more complex units can be developed before their components are complete, since any such component is easily replaceable by a fixed value or a dummy function that gives an appropriate result for a set of test inputs. Thus, there is a natural partitioning of the complex system, and questions of "top down" or "bottom up" programming become irrelevant, because of the ease with which attention can be shifted between detail and overview.

The process of application design, or any other non-trivial programming task, can be viewed as an exercise in language design on several levels. At each level the functions and the objects they transform constitute the elements of the language. Going towards the user interface, the functions become more specialized, the data objects thereby become endowed with more and more personality suitable to the particular application, and the language becomes more special purpose. At all levels except perhaps the highest, where the application is often best served by menus or isolated keywords, it is advantageous to emulate the properties of the APL primitive functions as much as possible. Indeed, even at the user level, particularly in data base inquiry applications, there is a growing class of users who are comfortable with the syntax of functions with arguments and results.

Operators

An operator in APL is a function that produces a new function, called a "derived function", which has all the properties of a primitive function. Operators make it possible to concisely represent procedures which, though conceptually simple, usually require elaborate programs for their expression in other languages. From the point of view of machine architecture, operators can be thought of as dynamically specifying the organization of a machine so as to efficiently accomplish a particular task. Conversely, it suggests that a machine with data flow or pipeline architecture might be more effectively programmed with a language rich in operators.

Four operators have so far been introduced in APL: reduction, scan, inner and outer product, and axis (Ref. 4). These are all structural, their derived functions applying the operator's function argument in defined ways along certain directions in their array arguments. The same end results could be accomplished without the use of operators, by the successive application of the appropriate functions to an array. One advantage of operators is that they obviate explicit iteration in very many common situations, reducing the probability of programming errors, while at the same time making it easier to produce optimized code in interpretation or compilation. A less obvious, but no less important, advantage is that they perform transformations on the functions rather than on the arrays, thus eliminating intermediate results that are wasteful of both time and space.

Although the operators in APL so far are all structural, this is not a necessary condition, and other kinds have been proposed, such as function composition, duality, derivative, and others (Ref. 5). Operators introduce a new dimension in programming, making functions objects of the language. They are a demonstrated way to make the break from conventional programming languages and the "von Neumann style" which the industry is now being exhorted to consider (Ref. 6).

Named Objects

While the use of names is nearly universal among programming languages, most languages have built into them the assumption that there is a "real", numerically addressable, memory underlying everything, and make provision for its explicit management. APL does not admit the presence of such a memory (except insofar as it is possible to address part of an array by numerical indices), and concerns itself solely with associating a name with an array or a function. Because of this uncompromising position, APL deals only with "virtual" memory — one that is independent of physical location — and it is a useful model for studying systems embodying such memories.

Most paging systems, as well as memory management systems that automatically stage data in a hierarchy of memories, perform their function with no reference to the structure of the data being manipulated, since such information is not readily available. Excessive paging and other kinds of inefficiencies can be traced to this lack. A system that recognized objects by name only would potentially have better performance characteristics, because even though some objects might be too large to move as a unit, the system would have the necessary information to handle the data movement more intelligently.

The advantages of named objects in application programming are well understood, and the use of names is another important factor in the productivity associated with APL. Full realization of these advantages, however, requires that the names be explicitly available as function arguments, a condition that does not obtain in APL systems when files are being used. This is a point to be considered in further development of the language itself. The problem is to retain linguistic facility and simplicity of syntax while dealing with objects that can be shared, since shareability is the one property of files that distinguishes them from other data objects.

Shared Variables

All systems are composed of processors that communicate with each other, and the means of communication between two processors is always one or more shared variables, that is, objects whose state can at least be set by one processor and sensed by the other. This fact is often neglected or overlooked in the design of information processing systems; the existence of the shared variable is taken for granted and attention is focussed instead on functions that manipulate it. As has been pointed out (Ref. 7), this is often convenient, but it can also be costly.

Many aspects of computing systems are best analyzed in terms of cooperating concurrent processors, even though their implementation may appear to be otherwise. Analysis in these terms will sometimes show that seemingly unrelated concepts are the same in principle.

For example, although an interrupt system is usually considered to be a subsystem of a single processor, in fact it uses relatively simple special-purpose processors to monitor conditions until such time as one of them recognizes a looked-for event, when it will set a variable (one bit, perhaps) it shares with a relatively complex general-purpose processor, which then acts at its convenience to do what the situation calls for. It can be seen that if the simple monitor processors were to be made more elaborate they could do more on their own, communicating with the other processor less frequently, using shared variables of larger capacity (several bytes, or perhaps a workspace), and thus evolve into a processing system with distributed intelligence.

The practical effectiveness of formalizing communication by explicit use of shared variables has been demonstrated in APL systems, and an APL machine design based upon the concept has been proposed (Ref. 8). However, the relevance of the idea is not limited to APL systems. In a strong sense, designing processors and designing languages are similar endeavors, since a processor can be identified with the language comprising the operations it is built to perform. When shared variables are used explicitly for communication, they cause the interfaces between processors or languages to be sharply defined, and thereby help to keep each processor or language focussed on its own specialization, uncluttered by concepts particular to others.

Workspaces

The user of an APL terminal has available an APL machine whose particular properties are determined by the active workspace. The workspace provides a default computing environment when no function is active, and maintains the global environment at all times. In particular, the workspace qualifies global names, isolating them from names in other workspaces, and thereby providing an automatic linkage mechanism for the subsystem represented by the objects in the workspace.

This linkage is unquestionably another important contributor to the productivity associated with APL systems. However, it does not solve all problems; when it is necessary to combine elements of different subsystems, for example, conscious effort is required to avoid name conflicts, precisely because the linkage is automatic. Nevertheless, workspaces and their management provide a model of a computing system that can deal effectively with a totally disparate work load.

Heretofore, the workspace organization has provided a shelter with respect to language design because of the absolute isolation it provides for objects in different workspaces. With growing recognition of the shortcomings of this absolutism, and the availability of implementations in which workspaces do not necessarily move in and out of active memory as a unit, the workspace idea now presents a challenge. It becomes necessary to consider the possibility of a linguistic facility for controlled access to objects in different workspaces, without sacrificing the present advantages.

Concluding Remarks

The interaction of the elements of APL technology with the fields of information processing listed earlier — machine architecture, system design, application design, and language design — can be displayed as a matrix. The analysis used here traversed the shorter dimension, with a heading for each of the six technological elements. It might have been done in the other direction, focussing on the four fields, to somewhat different effect.

Existing APL systems are useful in two ways. They are, first, powerful experimental tools, especially those systems that have a full shared variable facility which provides asynchronous communication between workspaces. Second, they can themselves be objects of study for the insights they may give into information processing; they provide a model with essential properties, complete, but relatively uncluttered as compared to conventional systems.

Acknowledgement

The ideas sketched here have developed over the years in discussions with my colleagues in the IBM APL groups and elsewhere. I am also grateful to K.E. Iverson and L.D. Misek for their careful reading and their comments on the manuscript.

References

1. A.D. Falkoff and K.E. Iverson, "The Design of APL", **IBM J. Res. Develop.**, Vol. 17, No. 4, July 1973.

2. A.D. Falkoff and K.E. Iverson, "The Evolution of APL", **Proceedings of a Conference on History of Programming Languages**, ACM SIGPLAN, June 1978.

3. E.J. Cannon, **Personal Communication**, IBM Corp., July 1977.

4. **APL Language**, Form No. GC-26-3847, IBM Corp., 1975.

5. K.E. Iverson, "Operators and Functions", **IBM Research Report**, No. 7091, April 1978.

6. J. Backus, "Can Programming be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs", **CACM**, Vol. 21, No. 8, August 1978.

7. A.D. Falkoff, "Some Implications of Shared Variables", **Formal Languages and Programming**, R. Aguilar, ed., North Holland Publishing Co., 1976.

8. R.H. Lathwell, "System Formulation and APL Shared Variables ", **IBM J. Res. Develop.**, Vol. 17, No. 4, July 1973.

THE EFFICIENT USE OF APL IN AN EDP ENVIRONMENT: A REAL TIME FINANCIAL INVENTORY CONTROL SYSTEM

Sandra L. Browne
Adrian A.J. Browne
A & S Financial Consultants Limited
Toronto, Ontario

Summary

There are few applications in the area of finance which are as technically demanding as a real time trading system. For an analyst they are difficult, but the real technical challenge is the requirement for machine reliability and the use of the computer environment to its fullest extent to provide the required flexibility and to minimise costs.

The application to be discussed in this paper was written for one of the major Canadian Chartered Banks to assist in the management of short term liabilities. These liabilities were made up of different types of money market paper issued by the Bank in amounts greater than $100,000. This system had to be capable of handling between 500 and 2,000 new issues made each day across Canada, and of course, an approximately equal number of daily maturities. With this volume of activity, there was a real necessity for meticulous procedures to ensure the integrity of the data base.

In order to fulfill the technical requirements imposed upon this system, it was necessary to use a broad spectrum of the technical features of SHARP APL. This paper discusses the factors which governed the choice of T-tasks, N-tasks, Shared Variables, B-tasks, HSPRINT output options and system integration with the Bank's own Data Centre. It is shown that further major improvements in cost effectiveness have been demonstrated in another system by introducing the use of micro computers for initial processing.

The Short Term Liability Issue Process

Each morning, the Chief Liability Trader is responsible for reviewing the current money market situation together with the Bank's total Short Term Liability Position, including the rate structure, term structure, and maturity schedule of that position. He then sets the card rates for both wholesale and retail short term instruments for the day. These card rates are set on the basis of the term to maturity of the paper to be issued, and also specify the call features to be attached to all paper which is issued.

Once these card rates have been set, they are sent to all branches of the Bank. The large Canadian Chartered Banks have 1,000 to 2,000 branches spread throughout the country. When a customer arranges to purchase a short term note with a face value over $100,000, the branch is required to notify a regional office with regard to the amount, term and the rate at which the issue was done. These regional offices are located at major commercial centres across Canada, usually Montreal, Toronto, Calgary and Vancouver.

In the past, the regional offices were responsible for telexing the information received from the branches to Head Office where it was entered into a computer system. This system produced reports on daily activity at 4:00 p.m. Toronto time, when entries were cut off to produce an activity summary for the Bank of Canada. This method of operation meant that the Chief Liability Trader was unable to accurately assess the change in his total position throughout the day. In addition, there was a significant time lag in the data as the cut off time for data entry was equivalent to 12:30 p.m. in Vancouver and 1:30 p.m. in Calgary.

In the old computer system there were few techniques used to ensure the integrity of the basic data base. Relatively few verification checks were performed on the initial data entry. and overall data base integrity was only roughly computed from a basic set of control totals. In addition. the method which was used to confirm that the information received by Head Office was correct was far from timely. Each branch was required to mail Head Office special confirmations which were copies of the contracts which had been issued during the day by that branch. When these confirmations arrived at Head Office about three days later, they were manually matched to the entries which had been made. Special enquiries then had to be made for those contracts for which there had been no entry and for those entries for which no contract had been received.

System Design Requirements

It was proposed that the new computer system should enable the Bank to put a terminal for direct system input into as many Regional Offices as was considered desirable. In addition, it was decided to replace the contracts sent by the branches to Head Office with special accounting forms for each piece of paper which had been issued during a day. These forms were to be received by the branches by the start of business the following day and were a source of information used by the Accounting Division to check the reasonableness of the interest accruals and payments reported by the branches on a monthly basis.

The following detailed requirements had to be met by the final system:

1. It had to be possible to have multiple input terminals at geographically remote areas. This meant that all input had to be self explanatory so that the system could be used by relatively untrained operators without technical support people available for back up.

2. All data entry had to be carefully checked and verified to ensure that the information being added to the main data base was correct. This meant that every precaution had to be taken to eliminate operator and trader errors and to ensure that complete restartability was available in case of any line problems or machine problems.

3. All errors which were found at the time of entry, or upon subsequent checking or after processing had added the information to the data base, had to be easily and cheaply corrected.

4. Complete real time information on the full position had to be maintained so that

the Liability Manager could obtain on-line enquiry reports on the rate and term structure of the total position on the activity to date during the day and on the current holdings of major customers.

5. The major computer processing integrating the new issues, eliminating issues which had matured, and producing reports which were required on a daily, weekly and monthly basis had to be done as cheaply as possible.

6. As the information contained in this system was confidential, these reports could not be produced on I.P. Sharp's line printer, so a tape was to be sent to the Bank's Data Centre. The report tape had to allow the Bank's Data Centre to direct the output to the appropriate Regional Data Centres for printing and delivery to offices and branches across the country.

The final operational configuration which was used is schematically shown in Figure 1 below.

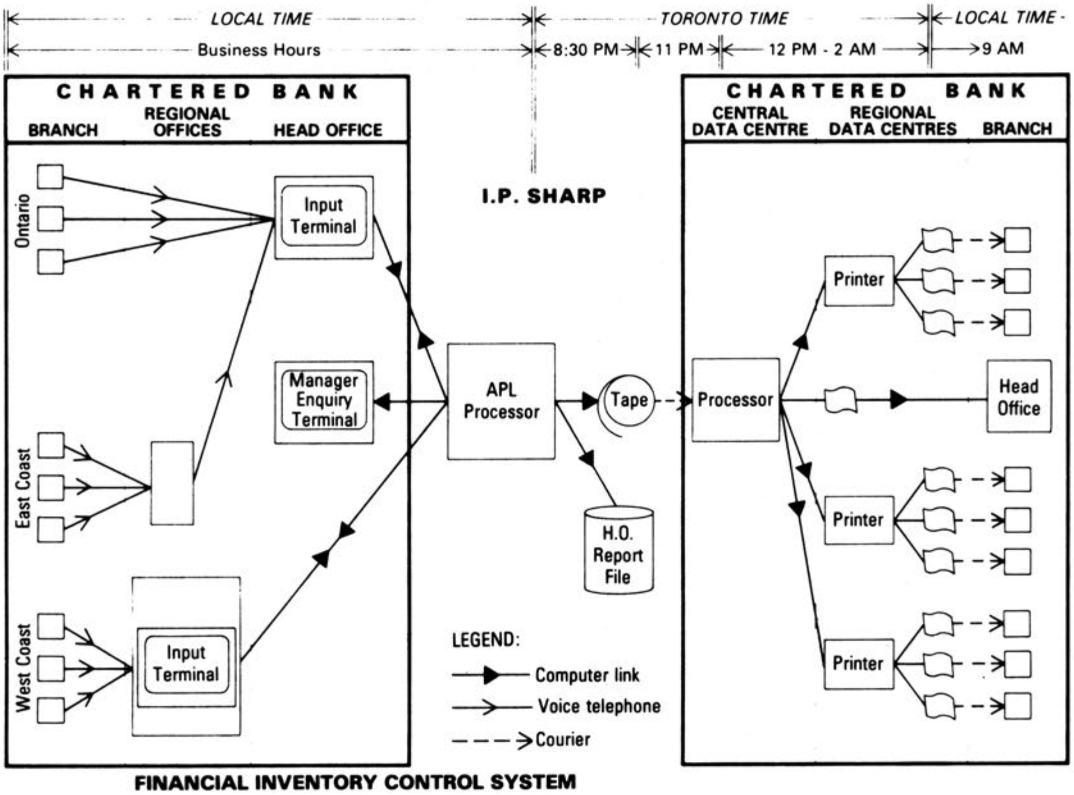


Figure 1

System Design and Implementation

Input

As anyone who has ever tried to maintain a data base is well aware, the most difficult part of data base management is trying to guarantee the integrity of the data. For this reason a great deal of emphasis was put on the immediate verification and continued rechecking of the data which went into this system.

The data entry programs were structured so that each piece of information which had to be entered by the operator was asked for separately. Hence, when the operator was entering a new note, 15 questions were asked to collect the 15 pieces of information required for that new note. This procedure meant that if at any time a new operator was unsure of exactly what information should be entered, it was possible to enter the data one item at a time. If the exact response to a question was unclear to the operator, an answer of SOS would produce a complete description of the required information. This procedure also ensured that any obvious error in entry could be rejected as it occurred and an explicit error message printed.

This type of entry procedure is ideal for anyone who is unfamiliar with a computer system, and who has no immediate recourse to technical support personnel. Ultimately, however, it is necessary to maximise the speed of entry which can be achieved by an operator who is familiar with the system. Hence, a buffered input approach was used for all data entry. Buffered input is a relatively simple technique in which the operator who is familiar with the system can answer in advance all questions which will be asked, separating each answer by an agreed upon delimiter, replacing 15 lines of input with one. If an obvious data error was found and rejected, the operator could correct the incorrect entry and complete the full input from the point at which the error was located or choose to start entering another ticket.

The sample terminal sessions shown in Figure 2 should help to clarify the advantages of using the buffered input technique for data entry.

Verification

When a data entry session was completed, the computer printed a series of blotters, one for each trader who had written out tickets used in that entry session. These blotters contained an expanded intelligible form of the information which had been entered. After all blotters had been printed, the operators would check to see if there were any obvious typing errors, and then the original tickets were returned to the trader together with the appropriate blotters for checking and correction.

Obvious errors, such as assigning a maturity date which falls on a national or local holiday, can be rejected at the time of entry. There are, however, many grey areas in any type of entry which cannot be automatically rejected. For this reason, when the blotter was prepared by the computer, warning messages were printed after each piece of paper in which it appeared there was any irregularity. Hence, a piece of paper which had been issued at a rate which was beyond the range of tolerance from the official card rate would be flagged as such for the trader's special attention.

```
                         BUFFERED INPUT ENTRY SESSIONS

QUESTION AND ANSWER SESSION

TRANSACTION NO.      : A34125
BRANCH TRANSIT       : 0314
PAPER TYPE           : TB
   ***    TB IS NOT A VALID PAPER TYPE
PAPER TYPE           : SOS

ENTER THE SECURITY CODE FOR THIS TRANSACTION. SELECT FROM : DR  BDN  GIC

PAPER TYPE           : BDN
AMOUNT               : 10000
RATE                 : 10.35
MATURITY DATE        : 10 10 80


BUFFERED ANSWER SESSION

TRANSACTION NO.      : A34125 ◇ 0314 ◇ TB  ◇10000 ◇ 10.35  ◇10 10 80
   ***    TB IS NOT A VALID PAPER TYPE
PAPER TYPE           : SOS

ENTER THE SECURITY CODE FOR THIS TRANSACTION. SELECT FROM : DR  BDN  GIC

PAPER TYPE           : BDN ◇10000  ◇10.35 ◇ 10 10 80


TERMINATED BUFFERED ANSWER SESSION

TRANSACTION NO.      : A34125  ◇ 0314 ◇ TB ◇ 10000 ◇ 10.35 ◇ 10 10 80
   ***    TB IS NOT A VALID PAPER TYPE
PAPER TYPE           : END

   REENTER TRANSACTION

TRANSACTION NO.      :
```

Figure 2

If no errors were found on the blotter, the trader simply initialled it. If, however, an error was found, the trader was required to mark the appropriate correction on the original ticket and return it to the operator for re-entry.

During overnight processing, a final consolidated blotter was produced for all entries made to the system during the course of the day. This was done for record and reference purposes and as a final check should any questions arise.

The final verification of the accuracy of the data was required at the branch level. When each branch received copies of the accounting forms for the paper which had been issued by it the previous day, it was required to verify that the details shown on the forms corresponded with those on their records.

Micro Computer Data Entry

The frustrating thing about verifying data entry is that it is both crucial to good data base management and expensive. It was for this reason that in a subsequent similar system which we wrote, we developed the use of a micro computer for initial data entry and checking. In order to be used in this way the micro computer must be one which can operate in tandem with the host time sharing system. As long as this type of relationship can be maintained, data entry can be done exclusively in the micro computer environment, together with the simple checking required.

There are times when it is necessary to access files which contain more information than can be held in core at one time in a micro computer. For example, the computer system discussed in this paper required millions of bytes of storage for the main data base. It is at this point that the micro computer's ability to communicate freely with the host computer system was of real importance. Those items which could only be verified using the larger, more efficient host time sharing system could be verified on a question and answer basis between the micro computer and the time sharing computer.

In the system in which we used this type of computer configuration, we found that input costs were cut by 75%, a saving well worth consideration for a system of this size.

There were two primary components which contributed to this saving. The first was the elimination of CPU cost for simple checking, and the second was the elimination of the character charges for printing the blotters. The only CPU and character charges incurred when using a micro computer to front end the system were those incurred for verification enquiries made to the host time sharing system and for the final transmission of the verified entries.

Shared Variable Data Base Manager

The one to one interface between the user and the computer is a straightforward if costly operation. When, however, there are multiple input terminals being used at the same time, it becomes difficult to ensure that the file updating process does not suffer from race conditions when multiple users are attempting to update the file simultaneously. Although the file hold operators on I.P. Sharp's file system can be used to try to ensure that two users are not updating the file simultaneously, they can be difficult to control in terms of restartability and file access privileges.

For this reason we chose to use a Shared Variable N-task as the Data Base Manager. This meant that regardless of how many users were entering data into the system, only one source of file updating was being used, as illustrated in Figure 3. In addition, file access privileges could be strictly controlled.

When the first user signed on for the day, the Shared Variable N-task was automatically started. It continued to run throughout the day until it was stopped by the overnight processing programs. Once the Data Base Manager N-task had been started, it made an offer to share to all user numbers within the system which were entitled to update the data base. This represented a simple mechanism for determining whether the Data Base Manager was in operation or not. If a user found the offer to share waiting for him when he signed on, the Data Base Manager was available. Otherwise the N-task would be started and input could continue.

DATA BASE MANAGEMENT SYSTEM CONFIGURATION

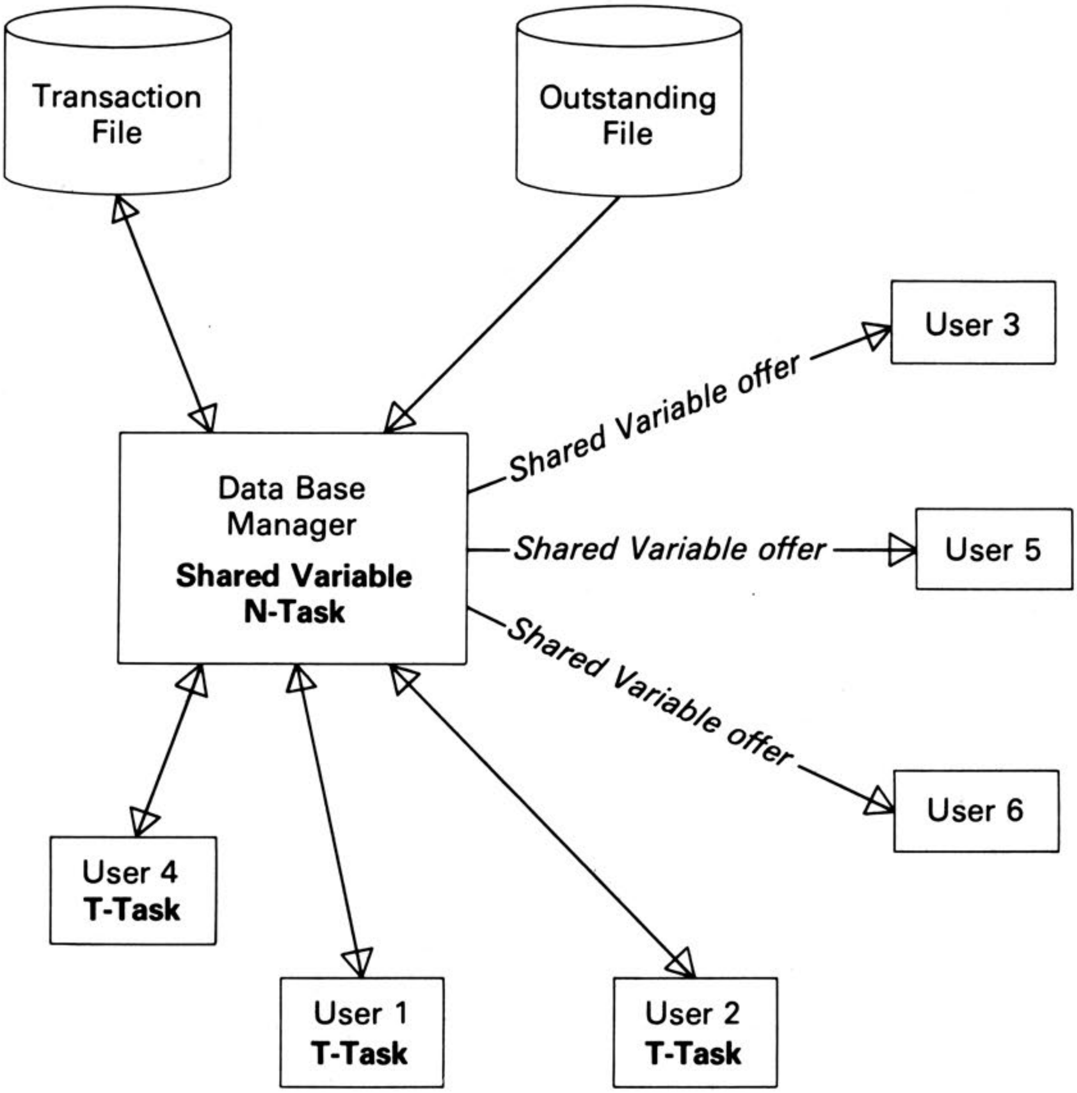


Figure 3

The interlock system available with shared variables makes it possible to ensure that only the information from one user at a time is being added to the file and once input has been collected and passed to the Data Base Manager for filing, there is no delay necessary while the file is updated. This means that there is a large amount of parallel processing possible between operator input and data base management.

As there is no guarantee that when the computer goes down the workspaces and file status with which it is restored are identical, the leading component on the file was a cross reference into all of the data components which followed. Each row in this cross reference component was tagged with the user number which had last entered or altered a component and a time stamp for the time at which it was done. In this way it was possible under restart conditions to tell the user which of his entries had last been permanently recorded in the system.

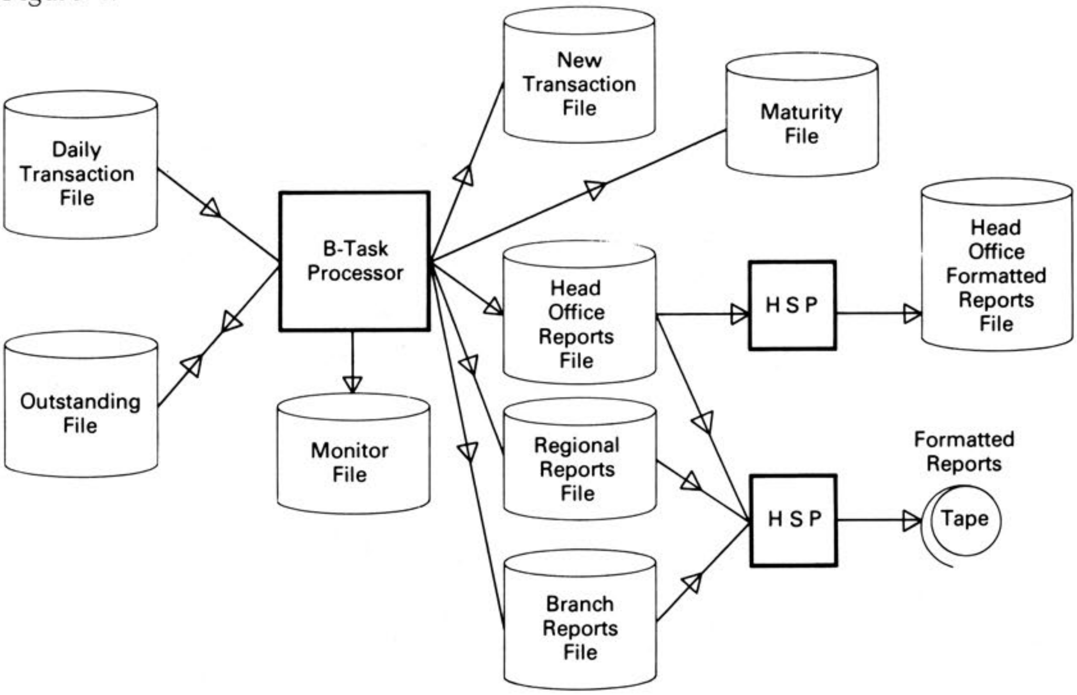
One of the main requirements for this system was that up-to-date reports on the current status of the position could be obtained. The use of a Shared Variable Data Base Manager facilitated the handling of these enquiries. The most up to date record of everything which had been done in the system during the course of the day always resided in the Shared Variable Workspace.

Overnight Processing

In order to minimise the cost of making alterations to data which had been entered either during the course of the day or previous days, a transaction file was maintained as a record of all entries of any type made during the day. Hence, if the user wished to alter the terms of an issue made on a previous day, the existing information was brought in from the outstandings file and a cancel and reissue added to the transaction file. Permanent changes to the main data base were only made during overnight processing.

The main computing requirement of this system was in the creation of the revised data base and in the production of reports. A B-task was used for automatic initiation at a set time of night to minimise processing costs and to ensure that processing was done after the close of business on the West Coast.

The operating configuration which was used for overnight processing is illustrated in Figure 4.



OVERNIGHT PROCESSING AND REPORT GENERATION

Figure 4

A basic principle followed in the overnight processing was to ensure that, should any machine problems occur during processing, the system would be completely restartable. A monitor file was continuously updated as the main processing program moved from step to step. In this way, a user who wished to monitor processing progress could do so without disturbing the B-task, or if any interruption should occur in processing, it would be relatively simple to determine what had caused the interruption of processing.

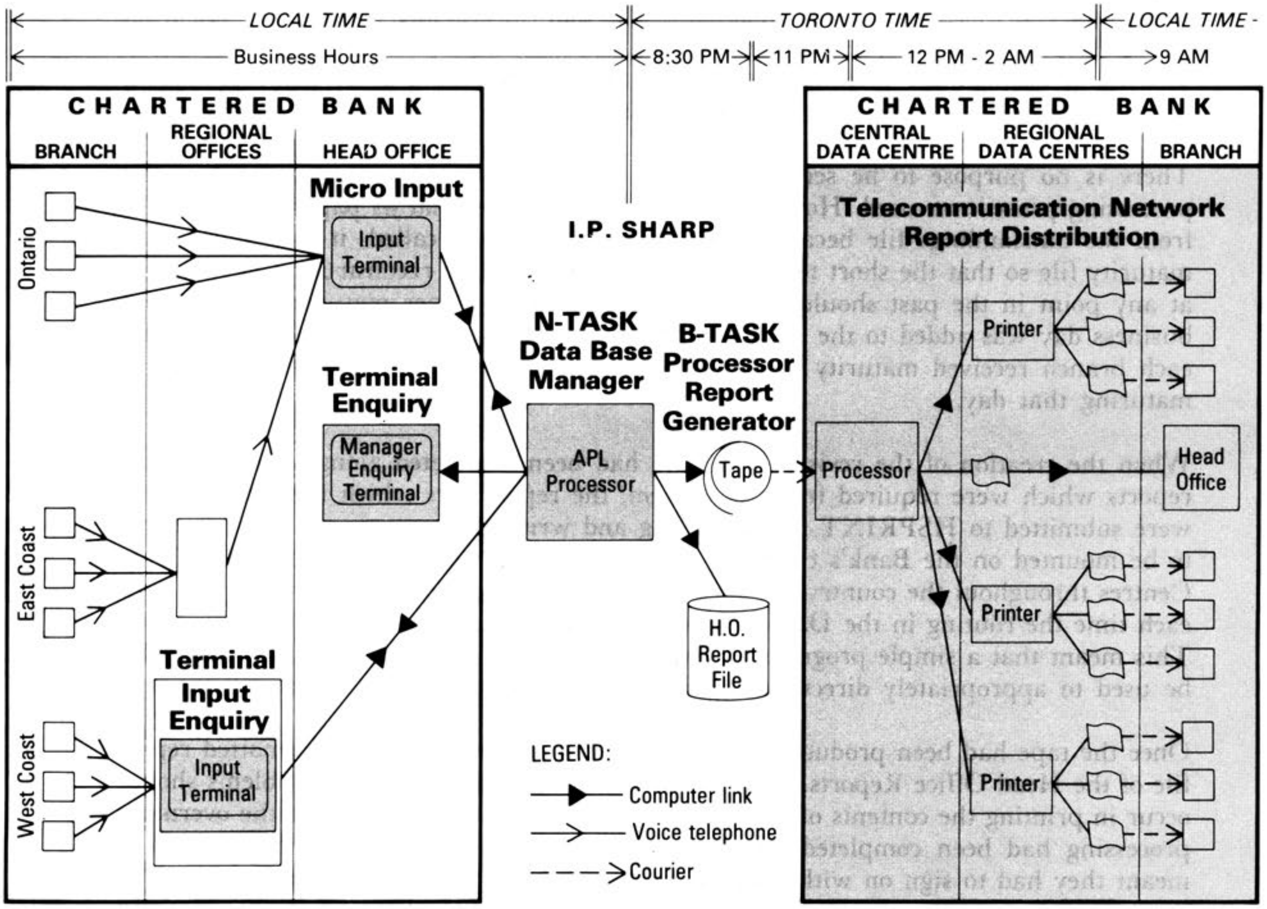
There is no purpose to be served by going into a detailed description of all of the processing procedures used. However, it is worthy to note, that as paper was removed from the outstandings file because it had matured or been called, it was added to a maturity file so that the short term liability position could be reconstructed or analyzed at any point in the past should it be desirable to do so. Paper maturing on the next business day was added to the branch report file so that at the beginning of each day, each branch received maturity reports summarising all of the paper which would be maturing that day.

When the creation of the revised data base had been completed along with all of the reports which were required from the system, the report files which had been created were submitted to HSPRINT for processing and writing on to tape. As this tape was to be mounted on the Bank's computer system to produce output at appropriate Data Centres throughout the country, an agreed upon header record was written on the tape each time the routing in the Data Centre communications network was to be changed. This meant that a simple program written by the Bank's Computer Department could be used to appropriately direct the reports which had been generated at I.P. Sharp.

Once the tape had been produced, HSPRINT was used to produce a formatted report file of the Head Office Reports. This file was kept on-line in case any problems should occur in printing the contents of the tape. This guaranteed that as long as the overnight processing had been completed, Head Office could always get its reports, even if it meant they had to sign on with a highspeed terminal to print them at their office. No similar precautions were taken with the other report files because it was deemed undesirable to print all 1,000 to 2,000 pages of reports which were produced every night on a terminal.

Conclusion

As illustrated in Figure 5, it is necessary to utilise a wide range of computer environments to effectively implement large scale applications. When this is done, however, it is possible to fully realise the advantages of each type of operating environment.



PRINCIPAL FEATURES OF FINANCIAL INVENTORY CONTROL SYSTEM

Figure 5

AN APL PROTOTYPE OF A MANAGEMENT AND CONTROL SYSTEM FOR A SEMICONDUCTOR FABRICATION FACILITY

Hassan Gomaa
General Electric Company
Syracuse, New York
and
Douglas Scott
I.P. Sharp Associates Limited
Toronto, Ontario

1. Introduction

One of the major problems in developing new computer applications is specifying the user's requirements in such a way that the specifications are correct, complete and unambiguous. Many data processing departments spend as much time maintaining existing computer systems as developing new ones, and many so called "maintenance" projects are in fact projects established to correct errors in the user requirements specification. Although prototyping is often considered too expensive, correcting ambiguities and misunderstandings at the specification stage is vastly cheaper than correcting a system after it has gone into production. This paper describes how an APL prototype was used to help specify the requirements of a computer system to manage and control a paperless semiconductor processing facility. The cost of developing and running the prototype was less than 10% of the total software development costs.

2. The User/Developer Communication Problem

The procedure usually followed in specifying user requirements is to produce a written specification in English which then has to be approved by the user. The main problems with such an approach are that the user often finds it dull to read such a specification and difficult to understand. Hence he is uncertain whether the proposed system satisfies his requirements. He finds it difficult to visualize how such a system will function in his environment.

The problem of communication between user and developer can be greatly alleviated if a prototype of the proposed system is developed. This is particularly the case if the system is to be used by a wide spectrum of users, all of whom have different requirements and some of whom are relatively unsophisticated with no previous experience of computers. It is difficult to ensure that the requirements of all these different classes of user have been met in a written specification.

With a prototype, users can exercise the prototype in the same way they would use the real system, and hence provide valuable feedback to the developers.

The prototype described in this paper was developed after a preliminary version of the requirement specification had been prepared. The objective was to allow users to exercise the prototype as early as possible in the software development cycle. Since the cost of making changes to a system increases dramatically as the development proceeds, it was desirable to obtain the user's feedback before the requirement specification was finalized.

Two methods were used to obtain the user's feedback: an evaluation form which they were asked to complete, and an on-line mechanism for entering (and reading) user comments. This allowed a user to enter his comments on-line, while using the prototype, and to review the comments of other users.

The system is a highly interactive one. Most user interactions with the system were explicitly prototyped. Some aspects of the system (such as interfacing with other computers) were not prototyped as it was felt that these requirements were generally clearly understood and would not benefit so much from prototyping.

3. Description of the Prototype

The main features of the prototype are described in this section. Some terms (process, recipe, operation, lot, work center) are defined. Menu processing, process management, lot tracking and data analysis are described.

3.1 Overview of the System

The system under discussion, called PROMIS, is a Process Management and Information System for a semiconductor processing facility, which is being developed jointly by I.P. Sharp Associates Limited and the General Electric Company.

Because this facility is a development laboratory, the process engineers need to modify frequently the fabrication processes that run in the facility. Using an iterative procedure, they experimentally determine the set of process steps that produce a semiconductor device with the correct performance characteristics and with a good yield factor. To meet this requirement the fabrication processes are all programmable, i.e. they are programmed into the system by the process engineers. To do this, the engineer requires no knowledge whatsoever of any programming language. He is provided with a series of simple English language prompts to which he replies with the appropriate information.

Integrated circuits are produced on silicon wafers, 3 or 4 inches in diameter. Depending on the complexity of the circuit and the wafer size, there may be 100 to 600 circuits produced on a single wafer. These wafers are processed in lots, and each lot contains typically 25 to 50 wafers.

The fabrication process for a typical integrated circuit consists of 60 to 100 process steps (see figure 1), and the processing for each step is controlled by process recipes maintained on a host computer and distributed to the appropriate work center when required.

These recipes usually contain 10 to 15 operations (see figure 2), i.e. 10 to 15 lines of instruction on a CRT terminal, and they may include:

- prerequisites, e.g. process gases required, special ambient conditions, etc.

- operations to be performed

- pre-defined set points for process control variables, e.g. temperatures, pressures, gas flow rates, etc.

- the duration of individual operations

- time constraints, e.g. the maximum elapsed time between two process steps

The duration of a process step is usually between 1/2 hour and 8 hours, and the operations that comprise that process step are performed within a single work center. The processing area includes about 16 work centers (e.g. wafer preparation, wafer clean, ion implantation, furnaces, photoresist, etc.) and each work center is equipped with a CRT terminal.

One of the difficulties associated with semiconductor processing is that process yields are severely affected by the intrusion of stray particles into the clean room, and for this reason no paper documents of any kind are allowed into this facility.

All the information display and data gathering functions required for lot processing must therefore be provided via the CRT terminal in each work center. Since there are no paper documents to accompany each lot through the facility, and since the computer system is the only mechanism for displaying process instructions to operating personnel, it is essential that the system be made "user friendly".

3.2 Overview of the Prototype

Four major parts of the system were selected for prototyping

1. Menu Processing

2. Process Management

3. Lot Tracking

4. Lot History Analysis

These parts were selected because they include the functions that will be most frequently used, i.e. they are all highly visible to the user.

In Part 1 (Menu Processing) the techniques for function selection were prototyped. Users select functions in the system by typing the name of a valid PROMIS command on the CRT keyboard. A complete list of all PROMIS commands may be obtained by typing HELP MENU, which lists the 3 levels of the PROMIS command tree (see figure 3).

During command selection, if the user hits RETURN without entering anything else, the system displays the list of commands that may be selected at that point in the command tree - i.e. the menu of available choices. If the user types QUIT during command selection, this action gets him back to the top of the PROMIS command tree.

To select a function from the menu, the user may type the entire command name (e.g. LOT PROCESSING), or starting with the first character, any unique substring of the command name (e.g. LOT or L). If the substring is not unique the system points out the ambiguity and reissues the prompt.

A novice user may work his way slowly down the command tree, entering the commands for each level one at a time. An experienced user may enter multiple commands on the same line, and may shorten each one to the minimum number of characters required to uniquely identify the command: for example, L T V means that LOT PROCESSING (L) will be selected from level 1, TRACK OPERATIONS (T) will be selected from level 2, and VIEW LOT RECIPE (V) will be selected from level 3 of the command tree.

In this way, the system was made friendly to both novice and experienced users, whose needs often conflict. Once a function has been selected, the user is presented with simple English language prompts for any data that is required to execute that function.

Part 2 (Process Management) is one of the two major interfaces between PROMIS and the process engineers (the other being Lot History Analysis). The objective of the prototype was to prove whether or not the concept of a programmable processing facility was workable - whether this approach could provide the engineers with all the tools they would need to produce any device in their facility. Executing this part of the prototype also generated a set of process, recipe and operation files that could be used in Lot Tracking.

Part 3 (Lot Tracking) is the key interface between PROMIS and the operators in the Programmable Processing Area. The objective here was to prototype the functions for tracking lots into work centers, viewing recipes, entering test results, and tracking lots out of work centers.

The procedure for executing a process step at a work center is as follows:

1. TRACK IN. A lot is selected from the wait list for that work center and is tracked into the work center, i.e. transferred to the active list.

2. VIEW LOT RECIPE. PROMIS automatically determines the recipe for that lot at that process step and displays it to the operator.

3. ENTER TEST RESULTS. If the recipe includes a test operation, the operator is prompted for each data item in turn, and the system performs range checks and pre-defined calculations on the test results.

4. TRACK OUT. The lot is tracked out of the work center, after which it appears in the wait list for the next work center.

For each process step executed, the prototype generated in the lot history file a complete record of what happened to the lot, including:

- date and time of TRACK IN
- date and time of TRACK OUT
- recipe executed
- work center id.
- equipment id.
- operator id.
- lot size at start of process step
- lot size at end of process step

- operator comment (optional)
- test results (if the recipe included a test operation)

Having collected a comprehensive record of what happened to each lot at each of perhaps 100 process steps, the problem of providing useful but easy access to that data is a major one.

In Part 4 (Lot History Analysis) the methods to be used for analyzing the lot history file were prototyped. Since the system is for a development laboratory, it has to support the kind of "witch hunting" analysis that process development engineers require: e.g. "give me a report on the test results obtained for device type 7-452, using recipe number 41260 version 3 and furnace number F91T2 or F91T3, and list the results for all such runs occurring between 15 January 1980 and 5 March 1980".

The challenge was to provide the considerable flexibility implied by a request of this type without obligating the user to learn a complicated report generation language. The solution adopted was as follows.

Considerable flexibility in the search criteria was provided by allowing engineers to specify the values of individual fields in the lot history file, using the field names.

However, in order to avoid the complexity of a report formatting language, the system provides a choice of 6 fixed report types, each report type providing a different level of detail: for example, overall process yields only, or yields for each recipe, or complete test results, etc.

The query language developed in the prototype allows process engineers to analyze process results

- by lot number
- by type of device
- by the process used
- by recipe
- by equipment used
- by type of wafer
- by date
- by manufacturers' batch number for the source wafers used
- by operator
- or by any combination of these

The following interactive sequence provides an example of using the query language to produce the report specified above: (user entries are underlined)

```
Terminal Input/Output                   Explanation

PROMIS COMMAND: O L R                   output data, lot history, reports
LOT HISTORY REQUEST FUNCTION            system prints name of selected function
REPORT TYPE: R                          R means results of test operations
ENTER SEARCH CRITERIA
FIELD NAME: DEV                         device type
KIND OF COMPARISON: EQ                  equals
VALUE: 7-452                            7-452 is the name of the device
MORE CRITERIA (Y/N): Y                  more criteria to be added? answer=yes
FIELD NAME: RECN                        recipe number

KIND OF COMPARISON: EQ                  equals
VALUE: 41260,3                          recipe 41260, version 3
MORE CRITERIA (Y/N): Y                  more criteria to be added? answer=yes
FIELD NAME; EQID                        equipment id.

KIND OF COMPARISON: IN                  in the list that follows
VALUE: F91T2 F91T3                      furnace 91, tubes 2 or 3
MORE CRITERIA: Y                        more criteria? answer=yes
FIELD NAME: SD                          start date and time
KIND OF COMPARISON: GE                  greater than or equal to
VALUE: 1980 1 15                        15 Jan 1980
MORE CRITERIA: Y
FIELD NAME: ED                          end date and time
KIND OF COMPARISON: LE                  less than or equal to
VALUE: 1980 3 5                         5 March 1980
MORE CRITERIA: N                        no means report request is complete
```

If the user forgets the report types available, or the field names or the kinds of comparison, hitting RETURN provides him with a list of all the options available followed by a repeat of the prompt. A description of the PROMIS lot history retrieval function is available to the user by typing HELP.

4. Experience with the Prototype

Users were given a two hour course and a practical demonstration of how to use the prototype. They were then given 2 weeks in which to experiment with the prototype and fill in a questionnaire with their comments.

Developing and using the prototype helped in specifying the user requirements in a number of ways. In software development projects, misunderstandings between developer and user tend to occur all too frequently, because of their different backgrounds. This type of misunderstanding was revealed on more than one occasion through use of the prototype.

Ambiguities and inconsistencies in the requirements specification were identified as the prototype was being developed. Omissions in the requirement specification were discovered by users who found that certain features did not provide all the information they required. They found other functions difficult or confusing to use. Some errors discovered in the prototype were actually found to be due to incorrect or missing requirements.

The requirement specification was revised extensively in the light of the experience gained in using the prototype.

Developing the prototype also provided valuable insight to the developers on how the system should be designed, in particular how the system should be structured, how files and data should be structured and which algorithms should be used.

5. Implementation Aspects of the Prototype

The prototype was developed by three people over a three month period using SHARP APL.

The total manpower spent on the prototype was 7 man months, which represents about 6% of the total estimated development effort of 10 man years. In addition, approximately $14,000 was spent in time-sharing costs. This figure includes all time-sharing charges incurred in developing the prototype software and in running it. It also represents the full customer billing rates, undiscounted.

Users of SHARP APL have found that the software development effort is often as low as one fifth of the time required to develop the same software in other high level languages such as BASIC, COBOL or FORTRAN.

APL has often been criticized for resulting in unmaintainable code [3]. Whereas there is some justification for this criticism, it is possible to write readable programs in APL by observing a disciplined approach and a good programming style [4]. However, maintainability is by no means as important a consideration in a prototype as it is in a production system.

SHARP APL has a file system which is simple and easy to use. It also has a powerful report formatting utility, which facilitates writing and changing sample reports, a valuable capability in any prototyping exercise.

System backup and recovery procedures are built into the SHARP APL time-sharing system, so the user does not have to spend time developing and running such procedures.

The prototype system under discussion was evaluated by personnel in Connecticut, New York state, California and Toronto, using the Sharp Communications Network. A further (possibly questionable!) benefit of the network was that senior managers who wished to evaluate the prototype, but were too busy during normal office hours, were able to take a terminal home with them, dial up the system, and exercise the prototype in the peace and quiet of their own homes.

The Sharp Electronic Mail System was used throughout the development and refinement of the prototype, to resolve issues that arose, and to keep all members of the project team in Sharp and G.E. in touch on a daily basis.

Another advantage of using SHARP APL for this prototype, and one that probably applies in many cases, is that the hardware on which the final system would run could not be delivered until 9 months after the prototype was completed. Since prototyping must be done as early as possible in the software development cycle, the availability of a time-sharing system is often an asset.

6. Conclusions

This paper has described an APL prototype for a semiconductor fabrication facility. The cost of developing the prototype was less than 10% of the estimated cost of developing the production system. This demonstrates clearly that it can be an economical proposition to develop a prototype. It is believed that this is also true for many other systems. The feedback obtained from using the prototype was found to be extremely valuable and more than justified its development.

7. References

[1] W.P. Dodd, "Prototype Programs", IEEE Computer, February 1980

[2] D. Scott, "PROMIS - A Process Management and Information System for Wafer Processing Facilities", Proceedings of the Microelectronics Measurement Technology Seminar, March 1980.

[3] E. Dijkstra,. "The Humble Programmer", Communications ACM, 1972.

[4] B. Kernigham and W. Plauger, "The Elements of Programming Style", McGraw Hill, Second Edition, 1978.

TYPICAL I^2L PROCESS

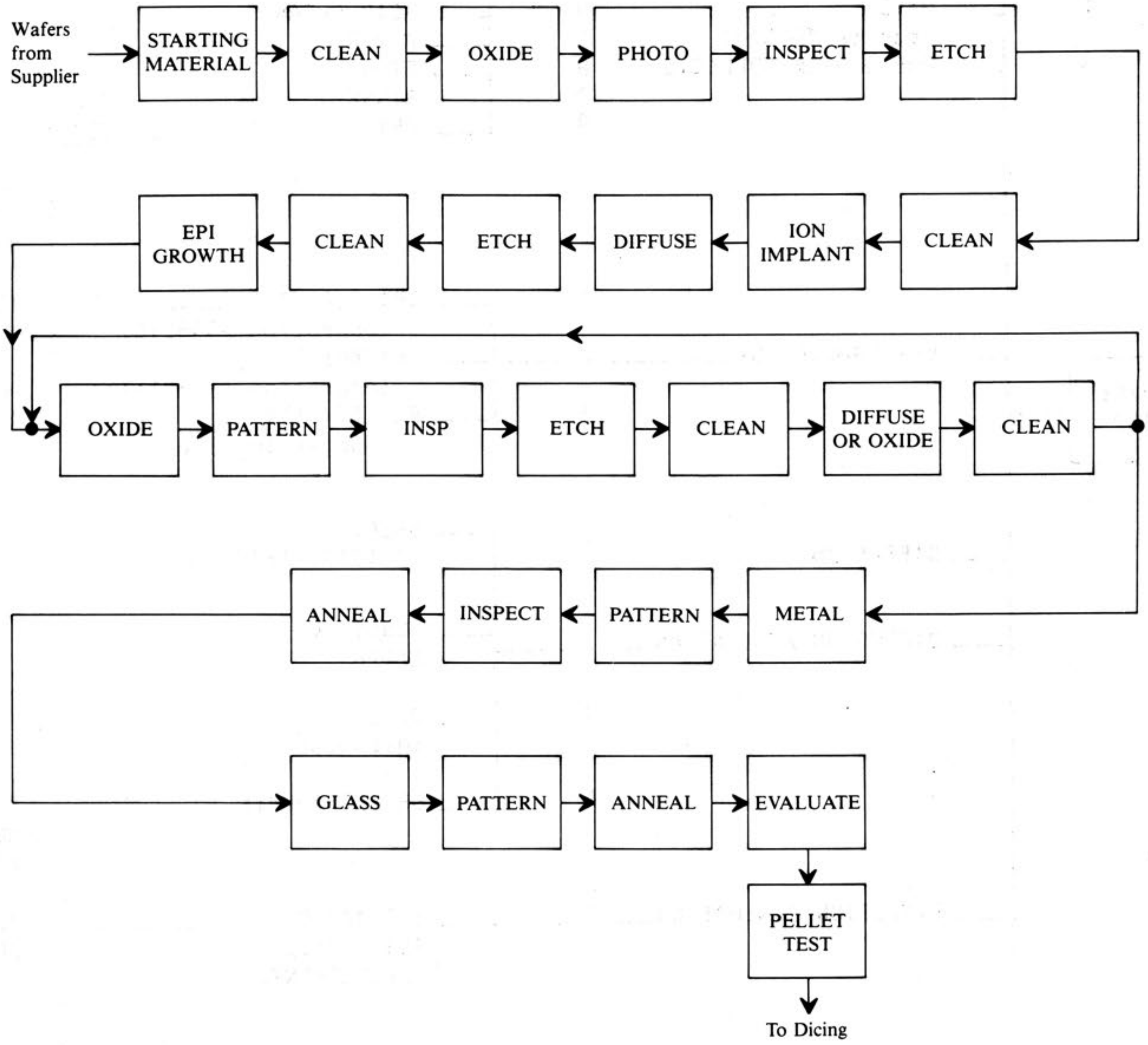


Figure 1

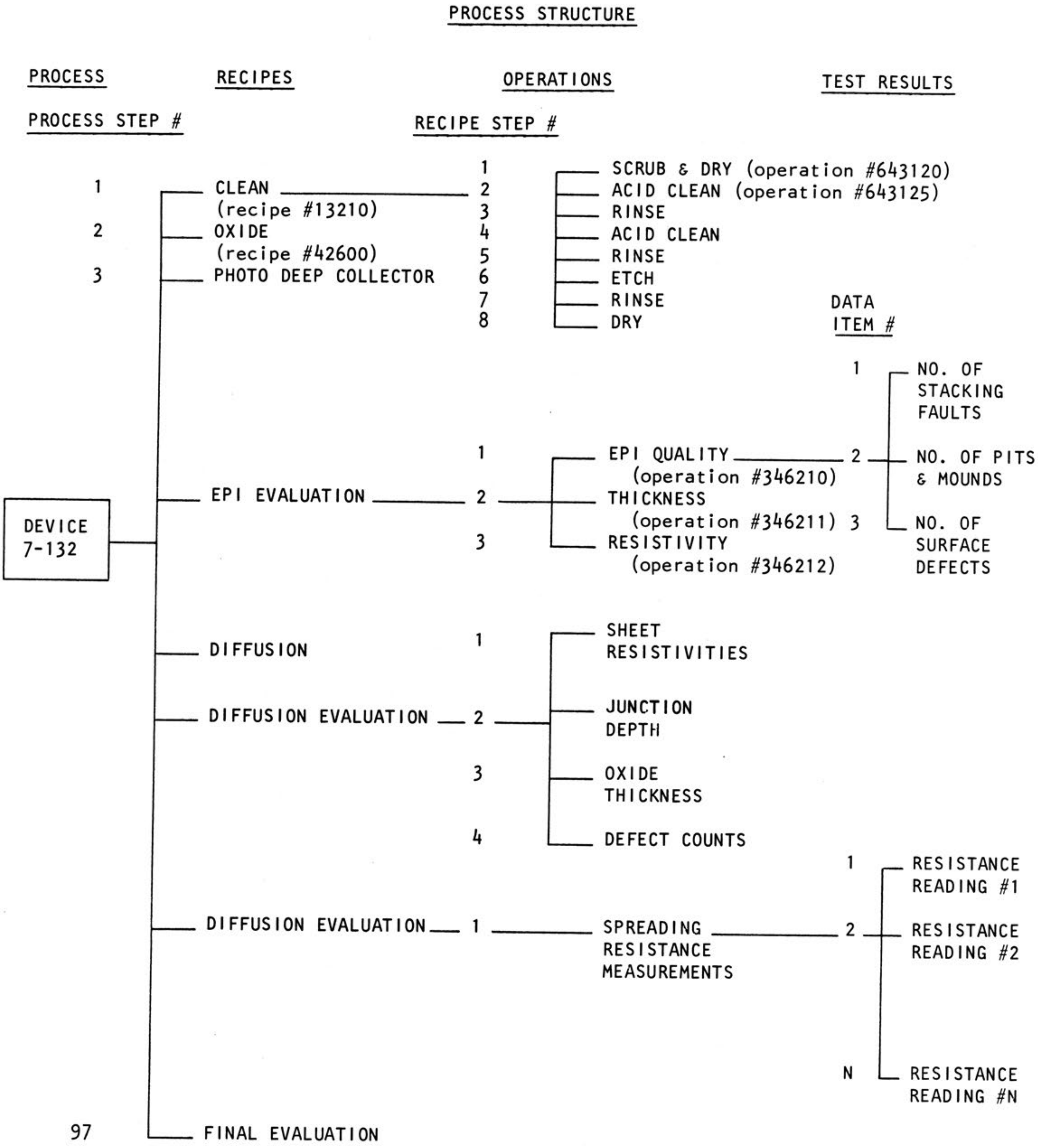

PROCESS STRUCTURE
PROCESS
RECIPES
OPERATIONS
TEST RESULTS
PROCESS STEP #
RECIPE STEP #
DEVICE 7-132
1
CLEAN
(recipe #13210)
2
OXIDE
(recipe #42600)
3
PHOTO DEEP COLLECTOR
1
SCRUB & DRY (operation #643120)
2
ACID CLEAN (operation #643125)
3
RINSE
4
ACID CLEAN
5
RINSE
6
ETCH
7
RINSE
8
DRY
DATA ITEM #
1
NO. OF STACKING FAULTS
2
NO. OF PITS & MOUNDS
3
NO. OF SURFACE DEFECTS
EPI EVALUATION
1
EPI QUALITY
(operation #346210)
2
THICKNESS
(operation #346211)
3
RESISTIVITY
(operation #346212)
DIFFUSION
1
SHEET RESISTIVITIES
DIFFUSION EVALUATION
2
JUNCTION DEPTH
3
OXIDE THICKNESS
4
DEFECT COUNTS
DIFFUSION EVALUATION
1
SPREADING RESISTANCE MEASUREMENTS
1
RESISTANCE READING #1
2
RESISTANCE READING #2
N
RESISTANCE READING #N
97
FINAL EVALUATION

Figure 2

PROMIS COMMAND TREE

LEVEL 1	LEVEL 2	LEVEL 3
PROCESS MANAGEMENT	PROCESS FUNCTIONS	• ADD PROCESS
		• CHANGE PROCESS
		• DELETE PROCESS
		• LIST PROCESS
		• FULL-LIST PROC
		• REC-LIST
		• F-REC-LIST
		• B-REC-LIST
		• RECIPE XREF
	RECIPE FUNCTIONS	• ADD RECIPE
		• CHANGE RECIPE
		• DELETE RECIPE
		• LIST RECIPE
		• F-REC-LIST
		• B-REC-LIST
		• RECIPE XREF
		• OPERATION XREF
	OPERATION FUNCTIONS	• ADD OPERATION
		• DELETE OPN
		• LIST OPN
		• FULL-LIST OPN
		• BRIEF-LIST OPN
		• OPERATION XREF
LOT PROCESSING	MANAGING LOTS	• START NEW LOT
		SPLIT A LOT
		REWORK LOT
		BACK UP LOT
		• DISPLAY A LOT
		• EDIT A LOT
	TRACK OPERATIONS	• INTO WORK CENTER
		• VIEW LOT RECIPE
		• OUT OF WORK CENTER
		• ATTACH TO WC
		• WORK CENTER DISPLAY
		• EQUIPMENT DISPLAY
		• FIND LOT
		• ENTER TEST RESULTS
	REPORTS	• LOT
		• FULL LOT
OUTPUT DATA	STATUS	• LOT
		EQUIPMENT
		• WORK CENTER
		• WAFER INVENTORY
		TANKS GAS
		BACK LOG
	LOT HISTORY	• DESCRIBE
		• REPORT
	OTHER HISTORY	ALARM HISTORY
		FACILITY HISTORY
MESSAGE	DIRECTORY	
	SEND	
	COMMENTS	• DESCRIBE
		• WRITE
		• READ
HELP	• DESCRIBE MENU	
	• SUMMARY	
ENTER DATA	FACILITY MONITOR	
	GAS TANK CHANGES	
	WAFER INVENTORY	• RECEIVE
		EDIT
		• DISPLAY
		• WITHDRAW
	SUPPLIER DATA	
	EMPLOYEE DATA	
FACILITY MONITOR	DESCRIPTION	
	LIST FUNCTIONS	
	REQUEST	
FUN AND GAMES		
BYE		

•=commands implemented in the prototype

Figure 3

SCHEDULING PRODUCTION WITH THE DYNAMIC LOT SIZE PACKAGE: APPROACH & METHOD

Andreas Buch
O.R. Research Projects
Zurich, Switzerland

Abstract

The production planner is confronted by the following problem situation:

1. to satisfy the delivery plan
2. to minimize the cost of production
3. to satisfy all capacity constraints

But already the basic question "when is what and how much to produce" is extremely troublesome. If several products compete for the same capacities, complex time, quantity, and cost interdependencies arise, making it one of the most demanding problems confronting management.

Numerous solution methods exist. However, they all suffer from major deficiencies: Requirements, capacities, and costs are assumed constant in time or several capacity constraints cannot be considered simultaneously. A solution using traditional methods oversimplifies the real problem or causes prohibitive cost.

The Dynamic Lot Size (DLS) approach and method will be discussed and examples referring to the use of APL will be given.

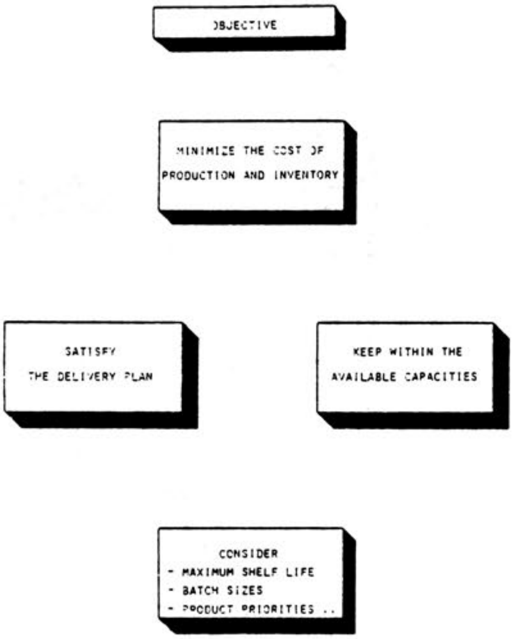
Problem Situation

In every production-inventory system there are economies of scale related to setup and inventory holding cost. The purpose of a lot-size model is to balance these costs in such a way that the total cost is minimized.

The problem would be very simple if the numerous real-world constraints associated with lot-size scheduling decisions could be ignored. Typical constraints are resource and deterioration, the first being upper bounds on resources available and the latter upper bounds on storage duration. Figure 1 summarizes the objective of lot-size scheduling.

In most traditional models, requirements, capacities, and cost are assumed constant in time. Thus the time dimension of lot-size scheduling is ignored (static methods). Such models often oversimplify real-world situations and may, if applied, cause more damage than good in practice. Figure 2 illustrates this oversimplification. The left-hand real-world situations all result in identical production schedules if a static model is applied. In a dynamic lot size environment, however, each real-world situation is considered exactly as it is.

Another aspect of traditional methods is their inability to consider constraints. If the three products in Figure 2 are all produced with the same equipment, it is very probable that the three optimal single-product solutions will not be feasible in practice. As a result the practitioner resigns and turns to his own rules of thumb from the very beginning.


OBJECTIVE
MINIMIZE THE COST OF PRODUCTION AND INVENTORY
SATISFY THE DELIVERY PLAN
KEEP WITHIN THE AVAILABLE CAPACITIES
CONSIDER
- MAXIMUM SHELF LIFE
- BATCH SIZES
- PRODUCT PRIORITIES ..

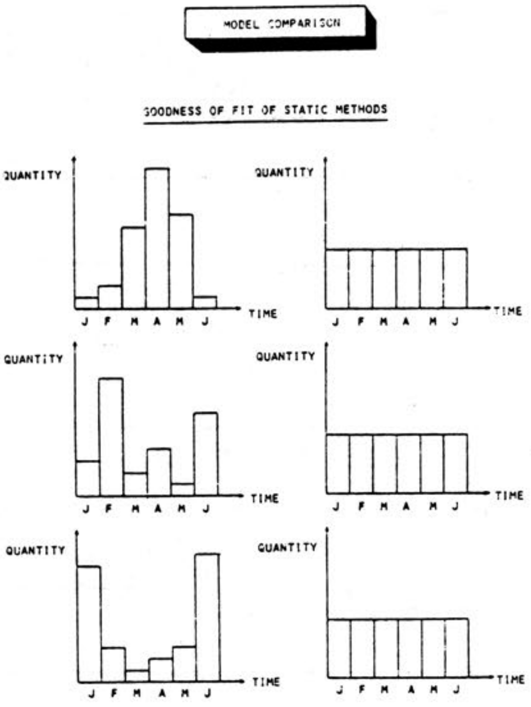


Figure 1 **Figure 2**

The three basic capacity considerations associated with traditional methods are sketched in Figure 3. In the first situation, capacities are fully utilized by always producing in large production runs. Thus the number of setups is minimized at the expense of large inventories. In the second situation (production to order) a new production run is always made when a demand occurs. This leads to a minimum investment in inventories, but may cause considerable setup cost. Further, the service level may suffer and the cost of eventual over-time work during the peak season is not considered. In the third situation, the seasonal peak demand is produced with the available resources immediately before the seasonal peak occurs. This approach may also be costly compared with a dynamic lot size solution.

So what are the alternative choices available to the production scheduler? In the first place he could make his production scheduling decisions using rules of thumb based on previous experience and intuition. Secondly, he could apply a selection of the infinite number of different square root formulas associated with static models, always risking oversimplification or wrong choice of formula. Finally, he could use a standard package for mixed integer programming for solving his problems. All alternatives are not very satisfactory.

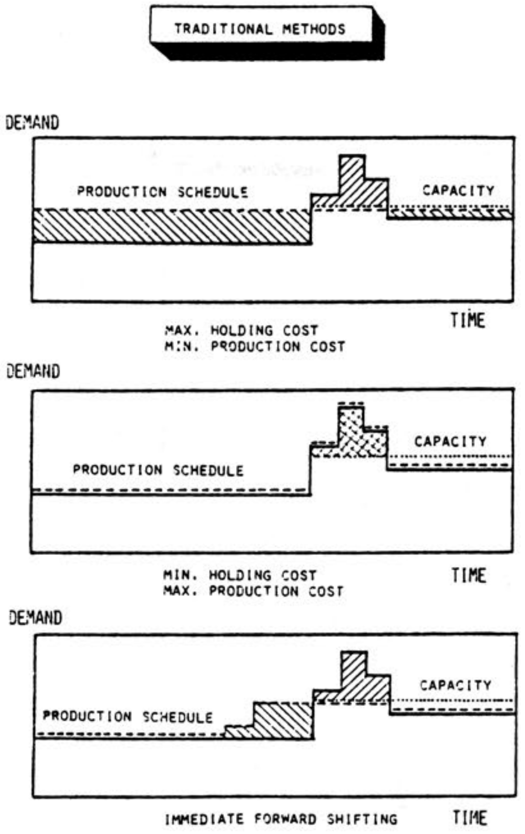


Figure 3

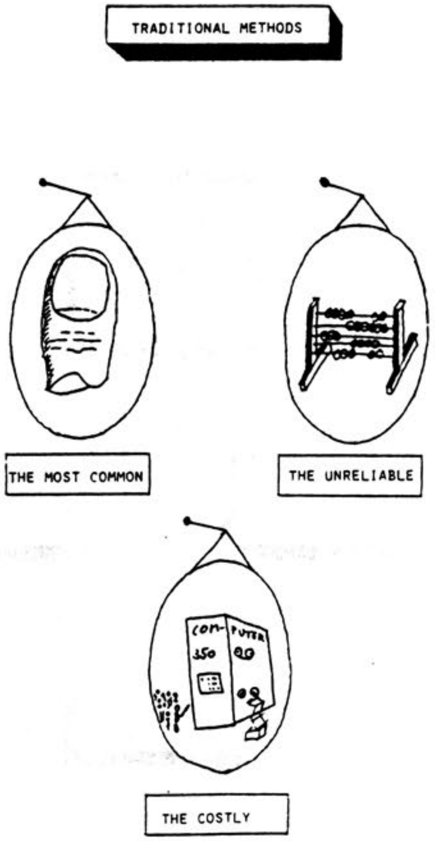


Figure 4

DLS Approach

The DLS approach is a modular approach. Each module is characterized by clear input/output specifications. In this way it is easy to know what one is talking about and the KISS (Keep It Simple ...) principle can be applied. In order to have some arrays to work on, the DLS input parameters are listed in Table 1.

The dimension of a DLS problem is defined by number of products times number of time periods times number of capacity constraints (see Figure 5). Fortunately, most real-world problems can be decomposed into several independent DLS problems since only seldom do all products compete for all capacities all the time. The decomposition question has to be carefully discussed with the practitioner in any DLS implementation.

Further, in a multi-stage environment it is important to find the right stages to schedule with DLS. The three typical multi-stage situations encountered in practice are indicated in Figure 6. Certainly, with proper definitions of unit and inventory holding cost, a chain of DLS modules can be used for handling those problems, the connecting links between the stages being bills of materials.

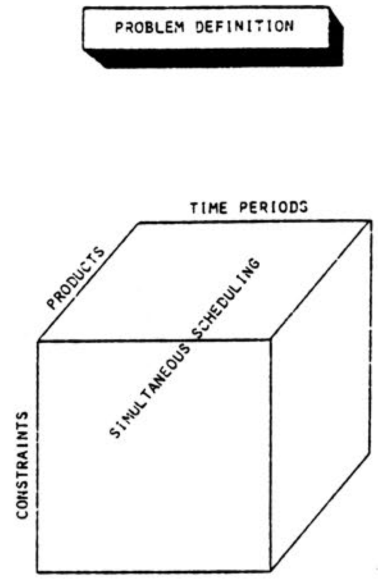


Figure 5

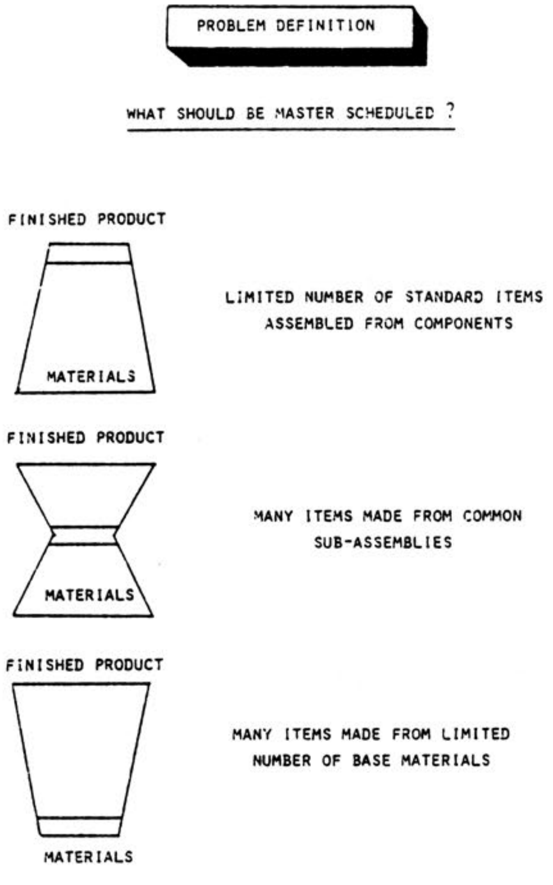

PROBLEM DEFINITION
WHAT SHOULD BE MASTER SCHEDULED ?
FINISHED PRODUCT
MATERIALS
LIMITED NUMBER OF STANDARD ITEMS ASSEMBLED FROM COMPONENTS
FINISHED PRODUCT
MATERIALS
MANY ITEMS MADE FROM COMMON SUB-ASSEMBLIES
FINISHED PRODUCT
MATERIALS
MANY ITEMS MADE FROM LIMITED NUMBER OF BASE MATERIALS

Figure 6

INPUT

PARAMETER	ρPARAMETER	DESCRIPTION	EXPLANATION
T		NUMBER OF PERIODS	TIME PERIODS TO BE CONSIDERED IN THE PLANNING HORIZON
P		NUMBER 'OF PRODUCTS	PRODUCTS COMPETING FOR THE SAME CAPACITY
C		NUMBER OF RESOURCES	CAPACITIES TO BE ALLOCATED TO THE PRODUCTS
G		NUMBER OF GROUPS	PRODUCT GROUPS TO BE CONSIDERED
RE	P,T	GROSS REQUIREMENTS	DELIVERY PLAN OR SALES BUDGET
SS	P,T	SAFETY STOCK	LOWER BOUND ON FUTURE STOCK LEVEL
II	P	INITIAL STOCK	STOCK ON HAND IN FIRST PERIOD
BS	P	BATCH SIZE	LOT SIZE ROUNDING PARAMETER
QD	P,T	QUANTITIES DISPOSED	FROZEN PRODUCTION QUANTITIES
CA	C,T	CAPACITY AVAILABLE	UPPER BOUND ON CAPACITY USAGE
GR	P,G	PRODUCT GROUPS	GROUPS FOR CONSIDERING JOINT SETUPS OF FACILITIES
ST	P,C	SETUP NEED	CAPACITY NEEDED FOR SETTING UP FACILITIES TO PRODUCE PRODUCTS
JST	G,C	JOINT SETUP NEED	CAPACITY NEEDED FOR SETTING UP FACILITIES TO PRODUCE GROUPS
CE	P,C	UNIT NEED	CAPACITY NEEDED FOR PRODUCING AN ADDITIONAL UNIT
SC	P,T	SETUP COST	COST INCURRED BY SETTING UP ALL FACILITIES TO PRODUCE PRODUCTS
JSC	G,T	JOINT SETUP COST	COST INCURRED BY SETTING UP ALL FACILITIES TO PRODUCE GROUPS
UC	P,T	UNIT COST	COST INCURRED BY PRODUCING AN ADDITIONAL UNIT
CC	P	CARRYING CHARGE	INVENTORY HOLDING COST PERCENTAGE
IR		INTEREST RATE	INTEREST RATE FOR CALCULATING PRESENT VALUE OF FUTURE COST
MC	P	MAXIMUM SHELF LIFE	DETERIORATION OR OBSOLESCENCE CONSTRAINTS
SU	∈0,ιP	PRODUCT PRIORITIES	DOMINATION OF RESOURCE ALLOCATION BY INSUFFICIENT CAPACITIES

Table 1

Now let us have a look at the smallest DLS component: the module for calculating net requirements. This module is of special interest, since the calculation involved is the most frequent calculation in inventory systems.

The problem situation is the following: Given gross requirements, safety stock, and initial stock, what are the net requirements over the planning horizon? The problem is solved using an APL idiom *ADJUST* for carrying negative numbers forward in an array. Since the scans involved in *ADJUST* are both associative, the calculation is very fast even for large arrays. Consider the following example with 5 products and 12 time periods:

```
      ∇ R←ADJUST R
[1]     R←0⌈⌈\+\R
[2]     R←R-(⍴R)↑0,R
      ∇


      ∇ R←NET∆REQUIREMENTS
[1]     R←SAFETY∆STOCK-(⍴SAFETY∆STOCK)↑INITIAL∆STOCK,SAFETY∆STOCK
[2]     R←ADJUST GROSS∆REQUIREMENTS+R
      ∇


        GROSS∆REQUIREMENTS
13  6 36 18 32 16  4  7  0  0 39 37
 7  0 13  1 16 26 19  0  0 17  0 12
18 30  0  0 30 17 18 13  0 12 25  0
 2 19  4  0  0  0  0  5 25 26  0 26
16 12 26 33  5  4  0 38 10  0 34  2


        SAFETY∆STOCK
10 10 10 15 15 15 20 20 20 25 25 25
25 25 25 20 20 20 15 15 15 10 10 10
 5  5  5  5  5  5 10 10 10 10 10 10
20 20 15 15 10 10 10 10 15 15 20 20
 0  0  0  5  5  5 10 10 10 20 20 20


        INITIAL∆STOCK
24 30 37 43 32


        NET∆REQUIREMENTS
 0  5 36 23 32 16  9  7  0  5 39 37
 2  0 13  0 12 26 14  0  0 12  0 12
 0 16  0  0 30 17 23 13  0 12 25  0
 0  0  0  0  0  0  0  0 27 26  5 26
 0  0 22 38  5  4  5 38 10 10 34  2
```

If net requirements must be divisible by certain batch sizes, we introduce the function *BATCH* in order to make the necessary shifts in the net requirements above. Note that this function is also based on the *ADJUST* idiom.

```
     ∇ R←BATCH NET;BATCHES
[1]    BATCHES←⍉(⌽⍴NET)⍴BATCH∆SIZE
[2]    NET←NET÷BATCHES
[3]    R←⌊+\(⌈NET)-NET
[4]    R←BATCHES×ADJUST(⌈NET)-R-(⍴R)↑0,R
     ∇


       BATCH∆SIZE
10 20 5 25 15


       BATCH NET∆REQUIREMENTS
 0 10 40 20 30 20 10  0  0 10 40 30
20  0  0  0 20 20 20  0  0  0  0 20
 0 20  0  0 30 15 25 10  0 15 25  0
 0  0  0  0  0  0  0 50 25  0 25
 0  0 30 30 15  0  0 45 15  0 45  0
```

And now we have reached the end of the basic module for calculating net requirements. The latter constitute input to the DLS mathematics to be described in the next chapter.

Another very frequent calculation in production-inventory systems is the calculation of capacity need for a specific production schedule. In order to illustrate how this is done in APL, we introduce another four matrices in addition to the ones previously introduced. Three capacity constraints are involved.

```
       SETUP∆NEED
 9  1 12
15 22  2
 0  6  0
11 11  7
 6  5  5
```

```
       GROUPS
1 0
1 0
1 1
1 1
0 1
```

```
       UNIT∆NEED
4 4 4
4 3 3
0 8 0
4 2 2
8 4 6
```

```
       JOINT∆SETUP∆NEED
41  2 27
25 48 38
```

As an example of joint setup, consider the production of instant coffee. Here a product is defined to be a certain glass containing a certain type of coffee. The products with the same glass form a group. Changing the glass would cause a joint setup. The DLS program allows any product to belong to any product group.

The capacity need for producing the production-to-order schedule can now be calculated.

```
      ∇ R←CAPACITY∆NEED SCHEDULE
[1]     R←(⍉SETUP∆NEED)+.×SCHEDULE>0
[2]     R←R+(⍉UNIT∆NEED)+.×SCHEDULE
[3]     R←R+(⍉JOINT∆SETUP∆NEED)+.×(⍉GROUPS)∨.∧SCHEDULE>0
      ∇
```

```
      CAPACITY∆NEED BATCH NET∆REQUIREMENTS
136 115 481 401 416 250 210 432 403 226 601 401
 84 257 336 256 564 339 379 321 226 278 602 314
 89 117 422 342 354 219 179 340 267 174 512 316
```

This calculation constitutes the main core of most scheduling packages in use. Very often the practitioner himself has to adjust his capacities to some production schedule or to make changes in his gross requirements and/or safety stock. It is the goal of DLS to avoid this.

The DLS output parameters are listed in Table 2.

In order to demonstrate the flexibility of the DLS module, consider the following simple example involving 10 products, 12 time periods, and 5 capacity constraints. Only 5 out of 17 DLS input parameters are activated.

```
      GROSS∆REQUIREMENTS
 0  0  0  0 21 44  0  0  0 39 16  0
19  0  0  0  0  9 35 10 46  6  0 49
 0  0  7  0  0  0 47  0  0  0 13  0
16 13 31  0  0  0  0  0 41  0  0 45
 0  0 39  0  0  0  0  0  0 29  0  0
 0 31 44 16  0 18 41  0 37  0  1 11
32 26  0 46 14  0 33 19 21 49 46 36
 0  4  2  0  0  8 38  0 23 37 22 31
21 25  0 39  3  0  0 22  0 17 19  0
42 37 40  5  0  0 49  0  0  0  2 39
```

```
      INITIAL∆STOCK
27 23 0 32 23 0 50 5 6 61


      NET∆REQUIREMENTS
 0  0  0  0  0 38  0  0  0 39 16  0
 0  0  0  0  0  5 35 10 46  6  0 49
 0  0  7  0  0  0 47  0  0  0 13  0
 0  0 28  0  0  0  0  0 41  0  0 45
 0  0 16  0  0  0  0  0  0 29  0  0
 0 31 44 16  0 18 41  0 37  0  1 11
 0  8  0 46 14  0 33 19 21 49 46 36
 0  0  1  0  0  8 38  0 23 37 22 31
15 25  0 39  3  0  0 22  0 17 19  0
 0 18 40  5  0  0 49  0  0  0  2 39


      CAPACITY∆AVAILABLE
820 630 790 750 650 650 720 590 590 530 670 550
620 690 620 520 710 680 770 530 830 540 570 680
710 770 620 630 750 670 720 670 760 740 480 610
590 690 800 820 660 630 640 770 540 870 590 590
720 560 760 660 740 770 550 730 700 630 520 680


      SETUP∆NEED
34 21 31 50  8
34 18 28 27 42
 1  0  0 28 46
14 49  0 43 35
23 41  2  9  0
36 22 30 38  0
35  0 38  9  1
25  4  9 46 40
38 22  1 30  0
 4 28 20 11 20


      UNIT∆NEED
3 6 6 3 1
3 5 1 8 4
3 0 0 7 3
3 1 0 3 6
3 2 1 6 0
3 6 3 2 0
3 0 8 2 5
3 2 5 1 5
3 8 1 2 0
3 5 5 7 3
```

OUTPUT

PARAMETER	*ρPARAMETER*	*DESCRIPTION*	*EXPLANATION*
Q0	*P,T*	*PRODUCTION SCHEDULE*	*OPTIMAL PRODUCTION QUANTITY FOR EACH PRODUCT IN EACH PERIOD*
PI	*P,T*	*STOCK ON HAND*	*RESULTING INVENTORY LEVEL OVER THE PLANNING HORIZON*
AD	*P,(T+1),T*	*AGE DISTRIBUTION*	*COMPOSITION OF PI USING THE FIRST-IN-FIRST-OUT RULE*
QR	*P,T*	*NET REQUIREMENTS REMOVED*	*REMOVED BECAUSE OF INSUFFICIENT CAPACITY*
RER	*P,T*	*GROSS REQUIREMENTS REMOVED*	*REMOVED BECAUSE OF INSUFFICIENT CAPACITY*
SSR	*P,T*	*SAFETY STOCK REDUCED*	*REDUCED BECAUSE OF INSUFFICIENT CAPACITY*
RC		*REST CAPACITY*	*UNUSED RESOURCES AFTER THE PRODUCTION OF Q0*
AC	*C,T*	*ADDITIONAL CAPACITY NEED*	*EXTRA CAPACITY NEEDED FOR THE PRODUCTION OF Q0+QR*
CO[1;;]	*P,T*	*SETUP COST*	*INCURRED BY Q0*
CO[2;;]	*P,T*	*UNIT COST*	*INCURRED BY Q0*
CO[3;;]	*P,T*	*HOLDING COST*	*INCURRED BY Q0*
<u>CO</u>	*G,T*	*JOINT SETUP COST*	*INCURRED BY Q0*

Table 2

Before running the DLS program it is often of interest to find out where capacity is insufficient to meet the production-to-order production schedule. This is given by

```
      0⌈(CAPACITY∆NEED NET∆REQUIREMENTS)-CAPACITY∆AVAILABLE
0   0   0   0   0   0 144   0  58 190   0 231
0   0  50   0   0   0  44   0   0  98   0  54
0   0   0   0   0   0 262   0   0 232 255 235
0   0   0   0   0   0 657   0 253   0   0 509
0   0   0   0   0   0 382   0  68   0   0 376
```

Program output with the above input data is

```
         PRODUCTION∆PLAN
 38   0   0   0   0   0   0   0   0  39  16   0
  0   0   0   0  50   0   0   0  46   6   0  49
  7   0   0  60   0   0   0   0   0   0   0   0
 28   0   0   0   0  86   0   0   0   0   0   0
  0  45   0   0   0   0   0   0   0   0   0   0
  0  31  60   0  59   0   0   0  37   0   1  11
 19   0   0  60   0   0  33  40   0  49  35  36
 47   0   0   0  26  27   0  60   0   0   0   0
 15  25   0  42   0  58   0   0   0   0   0   0
  0  20  40   9   0   0  79   0   3   0   0   2


         REST∆CAPACITY
211 166 450 159 150  60 345 230 258 145 409 147
 54   1  10  89  10   1 347 406 295 237 425 291
  1 454 190   4 326 467   3   3 510  11   2 114
 22  80 351  55   5 110   1 575   1 521 373   5
 33 480 620  86 328  44 127 189 445 271 320 235
```

In cases with no cost involved, like ours, the DLS default objective function is to minimize makespan.

However, by setting all (joint) setup cost to 1, a step in the direction of cost minimization can be made without any real-world cost data. This would simply mean that the number of (joint) setups is minimized.

Certainly, in order to profit fully by the DLS module, estimates of production and inventory holding cost should be entered from the very beginning. Then DLS solves lot-size and capacity scheduling problems simultaneously.

DLS Method

Basically, the DLS method is a dual gradient method, since it moves from infeasibility to feasibility using a gradient approach. In order to clarify the discussion, look at Illustration 1.

The mountains represent production and/or inventory holding cost. The mountain peaks are associated with expensive (joint) setups and/or inventories. Making a step means allocation of resources to one or several products, depending on the length of the step. Imagine the cost caused by a step to be the surface (or integral) of the cut associated with the step. Further, imagine each cut going down to a fixed level lower than the lowest point on any route from the START to the END line. We want to go from START to END causing a minimum of cut surface. The START line represents the situation where no capacity has yet been allocated, and the END line the situation where all products have been provided with the necessary capacity.

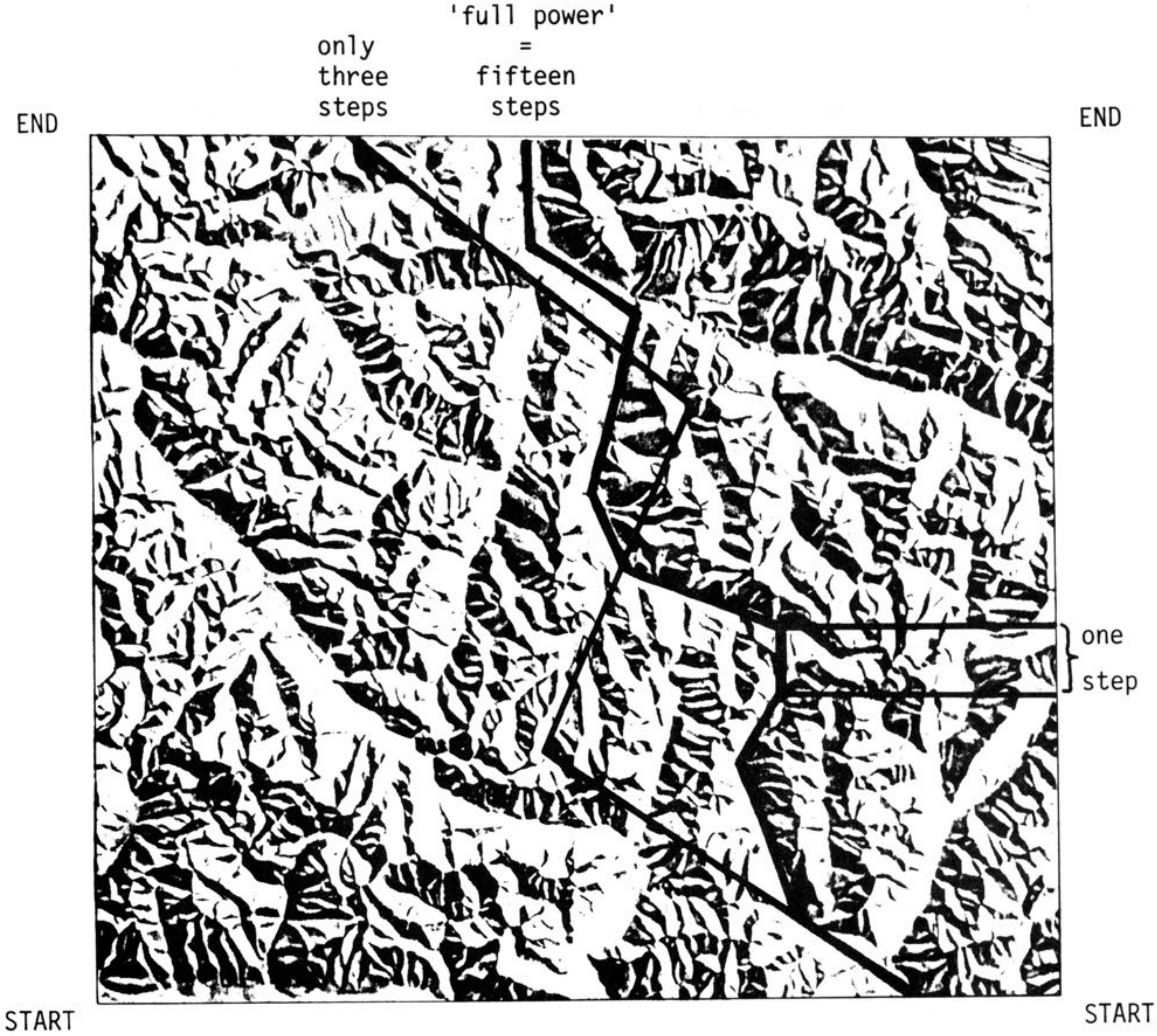


Illustration 1

Bearing this interpretation in mind, the basic DLS method can be explained as follows:

Stage 1: The START line represents the optimal single-product solution for each product considering no common capacity constraints. This solution is the optimal solution if no common constraints are involved. It is obtained using the standard dynamic programming technique from Bellman. In problems involving capacity constraints, this solution is very likely to be infeasible (see example in Figure 7 involving 100 products, 10 time periods, and 5 capacity constraints).

PROBLEM SOLUTION

PRODUCTION TO ORDER LOAD

	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	
FACILITY 1	60	72	99	61	106	98	135	81	99	117	%
FACILITY 2	71	71	101	92	87	86	121	84	90	110	%
FACILITY 3	41	57	76	64	93	104	69	92	108	110	%
FACILITY 4	66	73	101	72	98	121	107	97	113	139	%
FACILITY 5	46	49	98	72	85	79	82	94	84	112	%

INFEASIBLE + UNECONOMICAL

OPTIMAL SOLUTION LOAD WITHOUT CONSIDERING CONSTRAINTS

	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	
FACILITY 1	100	51	95	63	115	94	137	67	119	78	%
FACILITY 2	107	53	104	87	86	97	118	74	101	80	%
FACILITY 3	67	46	76	75	95	93	65	90	122	79	%
FACILITY 4	103	50	98	81	96	129	105	79	144	93	%
FACILITY 5	67	43	94	87	81	77	73	95	104	74	%

INFEASIBLE + ECONOMICAL

OPTIMAL SOLUTION LOAD CONSIDERING CONSTRAINTS

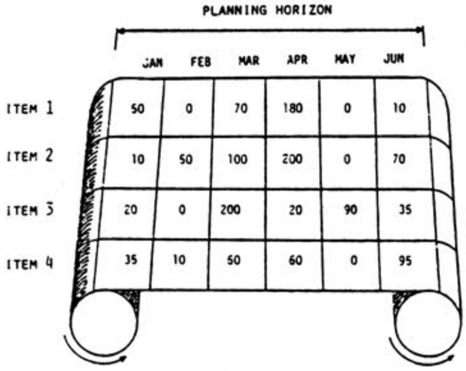
	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	
FACILITY 1	99	100	97	89	100	82	100	88	74	88	%
FACILITY 2	99	92	98	99	92	76	95	89	80	85	%
FACILITY 3	63	66	85	79	99	80	57	98	92	88	%
FACILITY 4	100	97	100	100	95	100	35	100	100	100	%
FACILITY 5	61	75	97	94	78	71	62	98	78	80	%

DLS SOLUTION: FEASIBLE + ECONOMICAL

Figure 7

ROLLING SCHEDULING

ROLLING HORIZON PROCEDURE



	JAN	FEB	MAR	APR	MAY	JUN
ITEM 1	50	0	70	180	0	10
ITEM 2	10	50	100	200	0	70
ITEM 3	20	0	200	20	90	35
ITEM 4	35	10	50	60	0	95

Figure 8

Stage 2: Now the direction of the next step must be set. For this purpose a quantity representing the potential savings resulting from allocating capacity to a product is calculated. This quantity has the following structure:

Minimum cost with no more bottle-neck capacity
in bottle-neck periods

Minimum cost with the capacity still available

Bottle-neck capacity needed in
bottle-neck periods

The latter quantity is weighted with relative bottle-neck importance in bottle-neck periods. Both cost quantities require the solution of separate dynamic programming problems. Therefore, at each step of the method

2 x number of remaining products

dynamic programming problems must be solved. Altogether, the number of dynamic programming problems solved in an optimization problem involving P products is P2(P+1).

Having calculated the above quantities, capacity is allocated to the product with the highest potential cost savings.

Stage 3: Having selected the specific product to be served with capacity next, the capacity allocation is done in a way as to secure sufficient capacity for the remaining products. The capacity needed for the production-to-order schedule plays an important role here (see chapter on DLS approach). If products still remain unserved with capacity, Stages 2 and 3 are repeated until no products are left.

Obviously, the DLS method involves a considerable amount of calculation. However, the APL formulation of the method has allowed for very acceptable CPU times. Almost no loops are involved.

In order to influence the CPU need, a step function is available to the user (see Illustration 1). With the step function the user himself can set the degree of exactness in the optimization. If he has a "flat" problem involving almost no cost "mountains", the direction of the gradient does not have to be modified very often. However, in problems involving all kinds of costly (joint) setups it is often advisable to use the "full power" DLS. Illustration 2 shows this situation. Standing in START it is very risky never to make any course correction since the END line cannot be seen or guessed. Certainly, the choice of the step length should also depend on the quality of cost input (see Table 1). If all cost is 0, one step is sufficient.

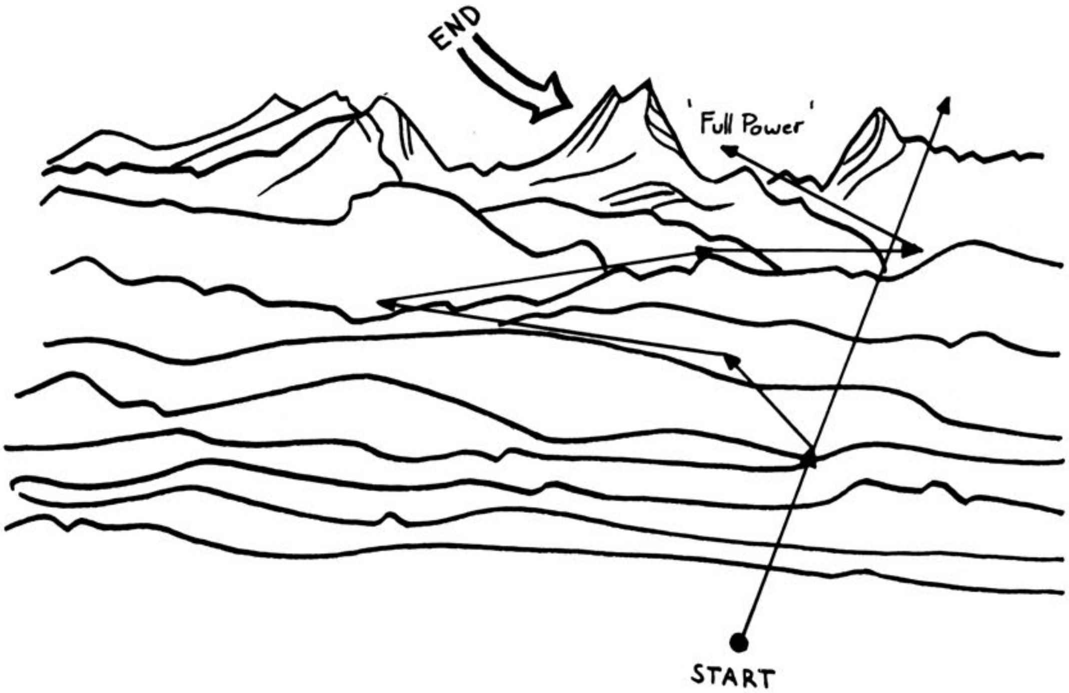


Illustration 2

The power of the DLS method can be demonstrated using the following three properties:

Property 1: There is a lower bound on the policy cost given by the cost of the unconstrained production schedule (from the START line). Thus, for "flat" problems it can be demonstrated with numerical examples that the unconstrained and constrained policy cost are almost identical.

Property 2: Tightening any capacity constraint must cause equal or increased policy cost. Thus, the monotonicity of the cost/capacity curve resulting from continuously tightening the capacity constraints must be monotonically increasing. Using the DLS method, numerical examples demonstrate the monotonicity of this curve.

Property 3: If, in large problems, the DLS solution is examined more closely, it appears that most products still have the optimal single-product solution. In the example contained in Figure 7, the number of optimal single-product solutions was 90 out of 100. And this even if the capacity constraints were very tight!

The CPU required using the DLS optimization module is a strictly concave function of the number of capacity constraints and a slightly more than linearly increasing function of the number of products as well as time periods. These properties are largely due to the use of APL. The WS need is relatively small.

The DLS optimization program is implemented in practice using a simple rolling horizon procedure illustrated in Figure 8.

DLS Benefit

What is the real benefit of the DLS optimization program to the practitioner? Experience has shown that this differs very much from case to case.

To most practitioners, however, the aspect of cost minimization plays an important role. The fact that DLS produces directly to real-world requirements (see Figure 2) guarantees cost-minimal (not minimal!) inventories without producing out-of-stock situations. On the contrary, the service level is improved. In order to account for uncertainty, KISS safety stock rules involving no sums of squares are used. DLS users don't need safety stock for hiding bad schedules!

Other production schedulers find it important to trace out future capacity bottle-necks and/or surpluses in order to be able to take corrective action in due course. Since the DLS module can consider any number of capacity constraints simultaneously over the planning horizon, it is a natural tool for capacity smoothing and forecasting.

Another benefit lies in the simulation capabilities of the model. As an example, consider the simulation of two different capacity levels associated with buying a new production facility or not. The differences in policy cost as well as capacity load are vital to the investment decision. Generally, by varying the DLS input parameters, the practitioner is able to do a substantial amount of relevant simulations.

To some producers (like pharma and food), the aspect of deterioration is vital. Since DLS can handle this kind of constraint, interesting analysis regarding cost of quality can also be made, tighter deterioration constraints being interpreted as an increase in quality.

DLS Interpretation

The class of constraints which can be handled with the DLS input framework is much broader than it may appear at first glance. The following examples illustrate this.

1. If we want to consider constraints on the number of setups in different time periods, this can be accomplished by introducing a capacity constraint where all products have setup need 1 and unit need 0. The capacity available would then be the maximum number of setups allowed.

2. Using the same idea, the number of joint setups can be considered. A new capacity constraint where each product group have joint setup 1 and each product setup and unit need 0 is introduced. Just like above, the corresponding capacity available would be the maximum number of joint setups allowed in the different time periods.

The DLS input parameter even allows for the definition of a transportation model. The necessary interpretations are contained in Table 3. The scope of the input parameters is much broader than in the traditional LP approach.

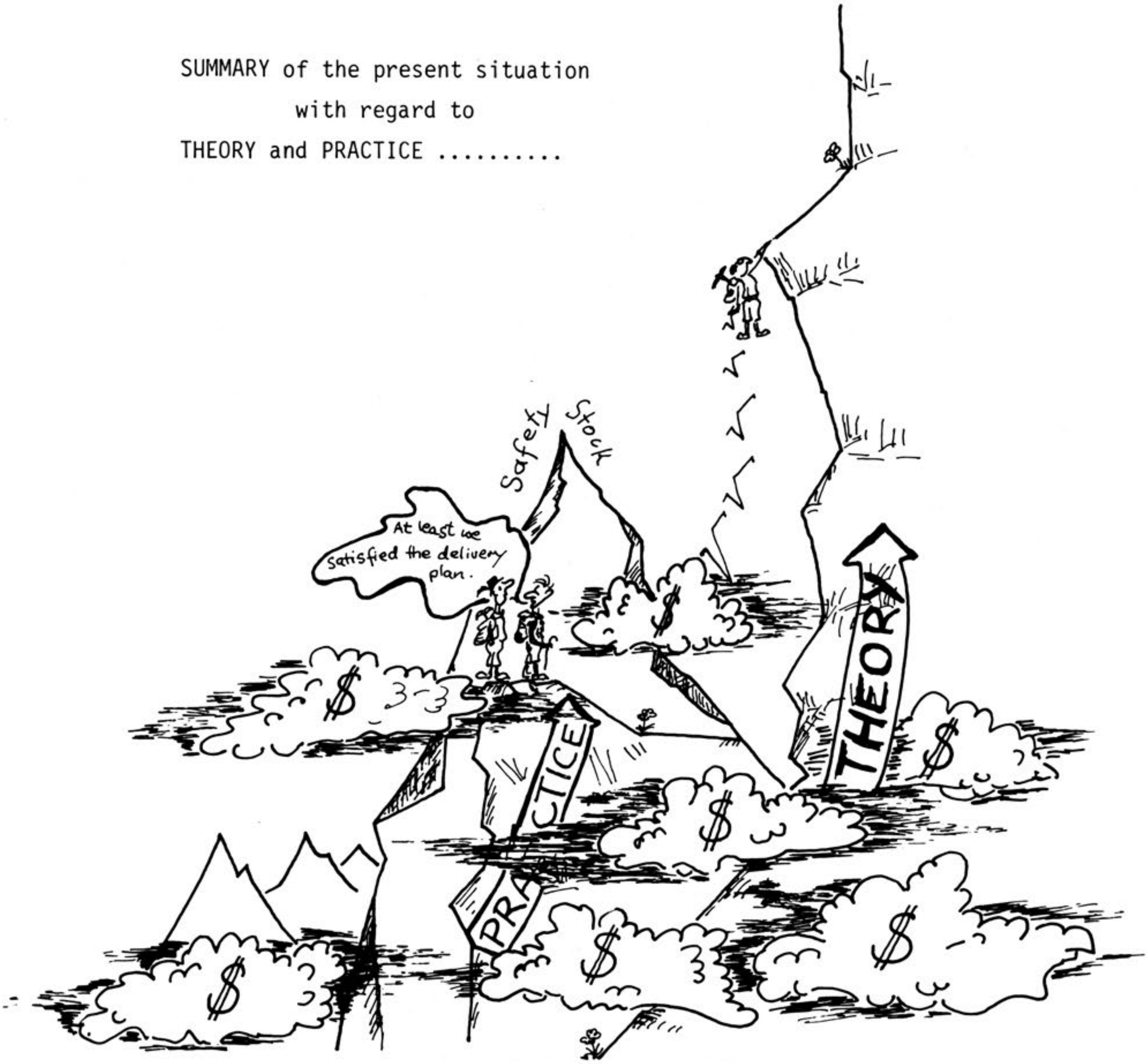
TRANSPORTATION

PARAMETER	*DESCRIPTION*
T	*NUMBER OF DESTINATIONS*
P	*NUMBER OF SOURCES*
C	1
RE	*QUANTITY AVAILABLE AT EACH SOURCE* (=*RE*[;*T*])
RS	*TRANSPORTATION QUANTITY ROUNDING PARAMETER*
QD	*FROZEN TRANSPORTATION QUANTITIES*
CA	*QUANTITY NEEDED AT EACH DESTINATION*
CE	(*P*,*C*)ρ1
SC	*SETUP TRANSPORTATION COST*
UC	*UNIT TRANSPORTATION COST*
SU	*PRIORITIES OF SOURCES OR DESTINATIONS*

Table 3

Appendix

SUMMARY of the present situation
with regard to
THEORY and PRACTICE ....



SOME RECENT DEVELOPMENTS IN ENERGY DEMAND MODELLING

G.C. Watkins
DataMetrics Limited
Calgary, Alberta

Introduction

My talk will be divided into five parts. First, I will discuss and make some comments on the nature of energy demand models, their role and purpose. Second, I will discuss the question of energy: output ratios and energy demand modelling. Third, I outline specifications of certain energy demand models, focussing, as the title of the talk indicates, on some new developments in this area. Fourth, to illustrate this new modelling work, I cite some recent results for the Canadian Manufacturing Sector. Fifth, I will make some concluding remarks, especially in the context of the use of computer support.

I. Nature of Energy Demand Models: Their Role and Purpose[1]

In essence, a model provides a framework for analysis, organizes data and establishes key relationships. As such, its purpose is to augment our capabilities to understand and assess problems. The accumulation of knowledge and the pursuit of the implications of that knowledge are the essence of sound analysis. Formal modelling makes the process explicit, visible and reproducible.

It is important to recognize that the assumptions on which a model rests also express its limitations. For example, models which are predicated on complete adjustment within one period cannot accommodate demands when contractual limitations inhibit flexibility; intrusion of a new source of energy supply is not included in models based on past data.

Although models can and do appear complex, they remain a highly simplified version of reality. Indeed, simplification is essential in modelling, to clarify basic causal mechanisms and to allow rapid analysis of inter-relationships:

> "Without simplification, the only model of reality is reality itself and only one experiment is permitted. With a model, many experiments are possible and unlike reality, we can take them or leave them."[2]

The application of a model must be guided by the context in which it is used. For example, co-linearity among a model's variables might be a problem for precise analysis

[1] The comments below in part echo those in [7].

[2] [7], p.6.

of causal relationships, but is less of a problem in a forecasting context. The reason why models can proliferate is that many approximations to the same reality are possible.

Often very simple aggregate analyses of an interaction can place the problem in perspective. Also, aggregate models can provide the conceptual background for the evolution of more sophisticated systems. As the potential deficiencies of simple models are identified and the assumptions relaxed, more detailed models develop and the range of application expands, a process inherent in some of the models discussed below. But a model is only an approximation to reality and must be judged accordingly.

Frequently, models extend the scope of the problem that can be studied if they accommodate inter-reactions which are not apparent at first glance. However, models cannot substitute for or replace the insight required by the analyst in determining the structure of the problem the model is intended to clarify.

In short, models are a tool to assist analysis; they do not replace judgement and detailed insights.

II. Energy: Output Ratios and the Modelling of Energy Demand(1)

The demand for energy is a derived demand in that it is transmitted from demands for goods and services which incorporate energy as an input. Trends in the ratio of energy consumption to the level of output — the so-called energy coefficient — are often used to examine energy demand in the industrial sectors.(2) The inference of this approach is that at a time of increasing energy prices, a rise in the energy coefficient is an indication of waste and inefficiency of a perverse price response. Correspondingly, a fall in the energy coefficient is evidence of the efficacy of the price mechanism and government regulations in enhancing energy conservation.

Work undertaken in the 1970's, of which the study by Berndt and Wood [4] is an early example, suggest that the use of trends in energy coefficients to analyse energy demand is too simplistic. Their work showed the importance of looking behind the energy-output coefficient to discern what is happening. Potentially important factors to explain variations in energy: economic activity ratios include the structure of the economy, energy prices, and prices of other inputs used in the production of goods and services. If these other components were significant, extrapolation of energy coefficients on some kind of trend basis would be appropriate only where relative energy prices and the structure of the economy remained constant. Both these hypotheses are tenuous.

A more comprehensive technique of analysis is to employ the economic theory of production, where energy is treated as one input among others which in combination produce output. Typically, the other inputs in addition to energy are physical capital, labour and non-energy intermediate materials.(3)

Thus, the methodology for investigating determinants of energy coefficients involves characterization of an economy or industry by a production and associated cost function which explicitly identifies energy as an input in the production of goods and services.

[1] For further discussion, see [12].

[2] For recent examples, see [8], [9], [10] and [13].

[3] Moreover, if the data permit, a breakdown of the capital stock between structures and machinery-equipment and labour between blue collar and white collar elements would be desirable.

This permits examination of what happens to energy: output ratios as economic incentives induce a variation in the mix of inputs required to produce a given level of output. A key question is whether inputs are substitutes or complements. For example, suppose the price of labour increased at a rate faster than the price of energy, all other input prices fixed. If energy and labour were substitutes, the relative increase in the price of labour to that of energy would tend to induce higher energy comsumption by substituting relatively cheap energy for more expensive labour. But if energy and labour were complements, the higher price of labour would tend to reduce both the demand for labour **and** the demand for energy. The estimation of relationship between factors of production according to whether they are substitutes or complements is a prime feature of the production and cost function approach.

However, in such analysis it is important to incorporate into the model the long-lived nature of typical energy-using equipment. When this is recognized, short-run energy price responses will generally tend to be smaller than long-run impacts. This requires a dynamic model specification, on which more below.

III. Model Specifications: Traditional and Recent

Early approaches to energy demand modelling in the industrial sector tended to treat energy demand simply as a function of energy prices and output. Such models were static and not grounded in any broader framework than that comprehended by simple price and output effects. By static I mean that no distinction is made between short and long-run adjustments of energy demand to energy price changes — all changes are complete within one year. Such an energy demand function could be written:

$$E_t = f(P_E, Q_t)$$

where E = energy demand

P_E = price of energy

Q_t = output

t = time

The problem with this kind of model is that while it is tractable and easy to estimate, it simply does not comprehend interrelations among factors of production in the industrial sector and is restricted by its static framework.

The static restriction is dropped in models which are inspired by what can be called the Balestra-Nerlove approach, which distinguishes between **flexible** and **captive** components of demand. The former is demand that is potentially variable because it is not committed to existing equipment; the latter refers to demand that is inflexible because it is tied to past investments. In this formulation we adjust energy consumption in the previous year to conform with current output by the ratio of current to previous output.(1) This adjustment is analogous to the way in which residential sector consumption is modified for temperature variations.(2) Thus we would write:

[1] For an application of this type of model to a utility company franchise area, see [5].

[2] For the latter, see [3].

$$E_t = f(P_E, I_t, Q_t, Q_{t-1}, E_{t-1})$$

where I = gross investment

t = time.

This formulation recognizes the distinction between short-run and long-run effects, but again is not explicitly related to any economic theory of production.

In the 1970's most of the more advanced work on industrial sector energy demand models used an approach which treated energy demand as one input among others in the production of output. Such models presumed firms acted to minimize costs of production and faced competitive markets for inputs. We can write the appropriate energy demand function as:

$$E_t = f(Q_t, P_K, P_L, P_E, P_M, t)$$

where P_K = price of capital

P_L = price of labour

P_M = price of intermediate materials

t = coefficient to designate technical change over time.

In this model, energy appears as one component of a production system, rather than as an explicit single equation specification. The model is derived from the economic theory of production, but is static.(1) Adjustment of all inputs to their long-run desired levels is assumed to be complete within one time period, e.g., one year.

The more recent developments in energy demand modelling to which the title of this talk alludes is the formulation of dynamic versions of the static production function type models I have just discussed. In a dynamic model, adjustments to changes in one period can take place over several periods. The distinctive feature of the dynamic versions of these models is the identification of what are called quasi-fixed inputs — an input which is fixed in the short-run but variable in the long-run; the other inputs are treated as variable. We can write the associated dynamic model as:

$$E_t = f(Q_t, P_K, P_L, P_E, P_M, K_{t-1}, t)$$

Again, energy is one component in a system, but the system is now dynamic, and short, intermediate and long-run effects are identified.(2)

IV. Some Results from Production-Cost Function Models

Both what I call the static and dynamic production function type models enable one to grope behind energy output ratios to discern what is going on, always assuming the models reasonably approximate the data they are trying to explain. As an example, let me summarize some results from static and dynamic formulations of production-cost function models for the Canadian total manufacturing sector, using data from 1957 to 1976. Here we distinguish four inputs: capital (K), labour (L), energy (E) and materials (M), which in combination produce gross output (Q).

[1] For details on the theory of this model, see [4].

[2] The theory is outlined in [1], [2] and [6].

As discussed beforehand, an important feature of production-cost function models is identification of whether inputs are substitutes or complements. The estimated relationships are shown in Table 1, for both static and dynamic models.

Capital and labour were estimated as substitutes in both the static and dynamic versions, but capital-energy relationships vary according to static and dynamic formulations, being substitutes in the former and complements in the latter. In the dynamic model, the capital stock was subdivided into structures and equipment. The relationship between structures and materials is complementary, while that between equipment and materials is competitive. This result is plausible, since structures and materials would tend to move in concert, while a firm may well enjoy a choice between buying finished goods as an intermediate input or self manufacture by investing in equipment. In the static model, capital remains aggregated and the overall relationship between capital and materials is complementary.

	Static	Dynamic
K-L*	S	S
K-E*	S	C
K-M*	C	C(ST), S(EQ)**
L-E	S-C	C(SR), S(LR)
L-M	S	S(SR), C(LR)
E-M	S	S

S = substitutes SR = short-run
C = complements LR = long-run

K = capital E = energy
L = labour M = materials

* Capital (K) relationship in dynamic models only relevant to long-run.

** ST = structures component of capital
EQ = equipment component of capital

**Inter-Factor Relationships, Static and Dynamic Models,
Canadian Manufacturing, 1957-1976**

Table 1

Labour and energy relationships are not uniform; in the static model they flip-flop between complementarity and substitutability, while in the dynamic model they are complementary in the short-run and substitutable in the long-run. This latter result assists in explaining the former, in that the static model results may well reflect varying proportions of short-run and long-run effects. But also the short-run—long-run distinction has some intuitive appeal: lack of flexibility in labour-equipment and energy relationships in the short-run[1] would tend to induce complementarity; greater flexibility in the long-run would tend to induce substitutability. Energy and materials are substitutes in both the static and dynamic models. Labour and materials are substitutes in the short-run but complements in the long-run.

[1] In the dynamic model, equipment and energy and complements.

Estimates of selected price elasticities and in particular energy price elasticities from both the static and dynamic production-cost function models are tabulated in Table 2. By way of definition, the own price elasticity of energy (ϵ_{EE}) is:

$$\epsilon_{EE} = \frac{\text{\% change in quantity of energy demanded}}{\text{\% change in price of energy}}$$

A cross price elasticity, say between labour demand and energy prices (ϵ_{LE}) is:

$$\epsilon_{LE} = \frac{\text{\% change in quantity of labour demanded}}{\text{\% change in price of energy}}$$

Typically, the sign of an "own" price elasticity will be negative: higher prices reduce consumption, **other factors held constant**. But the sign of a cross price elasticity will indicate whether inputs are competitive (i.e., substitutes) or complements. If the cross elasticity were positive, the two inputs would be substitutes; if negative, they would be complements. Thus, the signs of the cross elasticities in Table 2 correspond to the designation in Table 1 of whether inputs are substitutes or complements.

The own price elasticity of energy is, as expected, negative. For example, in 1976 the static model indicates energy consumption would fall by 3.2 percent if energy prices rose in real terms by 10 percent, all other elements — output, other input prices — held constant. The dynamic model results suggest that if in 1976 energy prices increased by 10 percent, by the end of that year energy consumption would fall by 1.9 percent and eventually it would fall by 4.8 percent. To take an example on cross elasticities: the cross price elasticity of energy prices on the demand for capital for the static model suggests that in 1966 a 10 percent increase in energy prices would increase the demand for capital marginally by .17 percent; in other words, minor components of capital would be substituted for energy. But in the dynamic model, capital and energy are complements, and in 1966 this model estimates that eventually a 10 percent increase in real energy prices would **reduce** the demand for capital by 1.8 percent. Analagous interpretations apply to the other coefficients shown in Table 2.

V. Concluding Remarks

The results I have tabulated demonstrate the way in which it is important in analyzing energy demand to look at the many factors which in combination with energy contribute to output. Such modelling can assist in explaining what at first glance may be a bizarre trend. For example, suppose the ratio of energy consumption to output increases at a time when energy prices are rising, seemingly a perverse result. Such a trend could be attributable to labour prices increasing at an even faster rate than energy prices, leading to the substitution of energy for labour. This does not mean energy price rises are not conserving energy. Instead, without such energy price rises, energy consumption might have been even higher.

Selected Price Elasticities for Energy, Static and Dynamic Models, Canadian Manufacturing

Total Manufacturing

		Dynamic		
	Static	**SR**	**LR**	
Own Price Elasticity of Energy				
1961	-.30	-.24	-.61	
1966	-.25	-.33	-.84	
1971	-.21	-.18	-.45	
1976	-.32	-.19	-.48	
Cross Price Elasticity Labour Prices on Energy Demand				
1961	.0006	-.17	1.55	
1966	-.0015	-.27	2.67	
1971	- 0026	-.14	1.78	
1976	-.0015	-.09	1.67	
Cross Price Elasticity Energy Prices on Labour Demand				
1961	.0078	-.02	.7	
1966	-.021	-.01	.09	
1971	-.037	-.01	.13	
1976	-.017	-.02	.35	
Cross Price Elasticity Materials Prices on Energy Demand				
1961	.222	.410	.474	
1966	.186	.606	.689	
1971	.154	.323	.368	
1976	.245	.276	.315	
Cross Price Elasticity Energy Prices on Materials Demand				
1961	.006	.014	.015	
1966	.005	.011	.011	
1971	.004	.008	.009	
1976	.007	.010	.010	
Cross Price Elasticity Energy Prices on Capital Demand			Equipment	Structures
1961	.018	-	-.015	-.172
1966	.017	-	-.004	-.176
1971	.016	-	-.006	-.133
1976	.018	-	-.007	-.161
Cross Price Elasticity Capital Prices on Energy Demand				
1961	.075	-	-.282	-1.131
1966	.082	-	-.405	-2.112
1971	.095	-	-.268	-1.428
1976	.088	-	-.213	-1.284

SR = short-run
LR = long-run

Table 2

Undoubtedly the effect of energy prices on energy consumption is not confined to one period. Consequently, a dynamic model which distinguishes between short-run and long-run impacts adds another dimension to the analysis.

While I have not in this paper actually listed the analytics of the production-cost function models to which I have referred, they are indeed complex and for elaboration I refer you to the references I have cited. At any rate, such models involve a system of equations, frequently non-linear in form. This means the estimation process is sometimes difficult and does require efficient non-linear algorithms and system of equation estimation procedures. Recently, some work we did provoked I.P. Sharp in Calgary to write some non-linear algorithms in APL (see [11]). Our own experience suggests that to achieve user efficiency it is important for users of such programs to closely liaise with the authors.

The use of production-cost function models — especially the dynamic ones — for policy simulation and projection requires the preparation of efficient simulation programs. The nature of the feedback in dynamic models entails quite complex procedures, which have not as yet been developed.

One final note is that the ability of APL to manipulate the morass of data required for estimation of time series and cross section models has been found helpful. For example, the manipulation of matrices which in other languages may require the writing of sub-routines can be accomplished with ease in APL.

List of References

[1] Berndt, E.R., M.A. Fuss and Leonard Waverman, "Empirical Analysis of Dynamic Adjustment Models of the Demand for Energy in U.S. Manufacturing Industries, 1947-74", Final Report, Palo Alto: Electric Power Research Institute, 1979.

[2] Berndt, E.R., C.J. Morrison and G.C. Watkins, "Dynamic Models of Energy Demand: An Assessment and Comparison", University of British Columbia, Department of Economics, **Resources Paper No. 49**, February 1980.

[3] Berndt, E.R. and G.C. Watkins, "Demand for Natural Gas: Residential and Commercial Markets in Ontario and British Columbia", **Canadian Journal of Economics**, Vol. X, No. 1, February 1977, pp. 97-111.

[4] Berndt, E.R. and David O. Wood, "Technology, Prices and the Derived Demand for Energy", **The Review of Economics and Statistics**, August 1975, pp. 59-68.

[5] DataMetrics Limited, "Price Elasticity of Apartment, Commercial and Industrial Demand for Natural Gas in The Consumers' Gas Company Franchise Area", Calgary, November 1979.

[6] Denny, Michael, M.A. Fuss and Leonard Waverman, **Energy and the Cost Structure of Canadian Manufacturing Industries**, Institute for Policy Analysis Technical Paper No. 12, University of Toronto, August 1979.

[7] Energy Modelling Forum, "Energy and the Economy", EMF Report 1, Vol. 1, Stanford University, September 1977.

[8] Kraft, John and A. Kraft, "On the Relationship Between Energy and GNP", **The Journal of Energy and Development**, Spring 1978, pp. 401-410.

[9] "Forecasts: The OECD Area to 1990?", **Oil and Energy Trends**, August 1979, pp. 12-15.

[10] Sakbani, M.M. and John J. VanBelle, "The Non-OPEC Oil Supply and Implications for OPEC's Control of the Market", **The Journal of Energy and Development**, Autumn 1976, pp. 76-85.

[11] I.P. Sharp Associates Limited, **Nonlinear Parameter Estimations**, 32 NPE, October 1978.

[12] Watkins, G.C. and E.R. Berndt, "Energy-Output Coefficients: Complex Realities Behind Simple Ratios", Paper presented to the International Association of Energy Economists Conference on International Energy Issues - The Next Ten Years, Cambridge, England, June 23-25, 1980.

[13] Wilson, Carroll J. (Project Director), **Energy: Global Prospects 1985-2000**. Report of the Workshop on Alternative Energy Strategies, Massachusetts Institute of Technology, New York: McGraw-Hill Book Company, 1977.

THE INTEGRATION OF APL INTO THE LARGER WORLD OF DATA PROCESSING

Eric B. Iverson
I.P. Sharp Associates Limited
Toronto, Ontario

Introduction

The general history of APL has been treated in a number of publications, notably **The Design of APL** [1], **The Evolution of APL** [2], and most recently **A Socio-technical History of APL** [3]. This paper concentrates on the evolution of the APL **system**: its origin as a stand-alone system, its integration into the general data processing environment, and speculation on its future.

APL started, and evolved, in relative isolation from the rest of data processing. This isolation benefited the language, and forced the development of a total environment that was uniquely appropriate. Together, language and environment make up an APL **system**. A key factor in the continued growth of APL usage is the proper integration of the APL system into the general data processing environment.

The First APL Time-sharing System

A time-sharing implementation of APL for the IBM 360 series of machines was first available to users in late 1966 at the IBM Research Center in Yorktown Heights, New York. When the project started, the IBM 360 hardware was quite new and there was no suitable operating system or software environment in which to embed APL. So the implementation was designed to be a complete, self-contained system that ran on a dedicated, stand-alone machine. Just as APL developed in relative isolation from other programming languages, the APL system developed in isolation from other systems. APL was not implemented within the constraints of a foreign software environment. Rather, the APL system, language and environment, was designed and implemented as a unified system by the same people who designed and used the language.

APL was an established notation before it was implemented. By the time the implementation project was started there was already an experienced user community. These users, plus valuable experience gained with earlier efforts (notably Herb Hellerman's PAT system for the IBM 1620 and Larry Breed's and Phil Abrams' IVSYS system for the IBM 7090) helped set the direction of the implementation.

The users of APL used it directly at their desk. This dictated direct access to the implementation, not a batch system. So a time-sharing system was developed, where the terminal became simply a more powerful pencil and paper.

The users knew APL and wanted to continue using it with the increased productivity allowed by machine execution. They were not interested in learning additional layers of a complex command language. So a very simple, but adequate command language was provided. The concept of a workspace was central to the entire system. It was easily understood by the user and was a natural way of organizing work in APL. It was the simplicity and elegance of the workspace that led directly to the simplicity of the command language.

The implementation was carried out with remarkably little compromise. The impossible was done again and again, yet the system was responsive and reliable. An example of integrity was the treatment of the character set. The importance of the rich set of symbols was fully recognized and a proper character set was provided. There was no compromise for an ad hoc, cumbersome, keyword system. Many later implementations, to their loss, made this compromise.

Complete control over the design and development of the entire system, and the extremely close cooperation (in fact hazy distinction) between the designers and the implementers, led to the development of a unique system. The Grace Murray Hopper award for "... setting new standards in simplicity, efficiency, reliability, and response time for interactive systems" given to Larry Breed, Richard Lathwell and Roger Moore in 1973, recognized the significance of their work.

Out of the Lab

The fact that this first implementation was a stand-alone system, requiring what at the time was a substantial amount of hardware, effectively limited its use to the IBM Research Center. The first step taken to get the implementation out of the lab was a modification so that the APL system could co-exist on the same machine as the DOS operating system. This resulted in the first APL program product, which was installed by several organizations, primarily universities, by early 1968. Shortly thereafter, a version that could co-exist with the MVT operating system was made available.

An important point in both these APL products was that the APL system, as seen by the user, was not compromised or changed because of idiosyncrasies in the operating system; the APL user could not tell if he was on a stand-alone, DOS, or MVT system.

The availability of APL in the DOS environment, and the flush of excitement about time-sharing in general, led to at least eight companies entering the APL time-sharing business in quick succession starting in late 1968.

On these early systems, the APL user was completely isolated from the rest of the data processing world. In fact, APL users were even quite isolated from each other. Although the command language was powerful, there was no way, under program control, to communicate outside of the active workspace. Moreover, it was impossible to get data into or out of the APL system other than through a low speed terminal.

The immediate and most serious problem faced by the commercial entrepreneurs was the inability to take in, process, and output the large amounts of data that are endemic in the business world. Most of these pioneers quickly implemented file systems that provided APL users convenient and efficient program access to arbitrarily large amounts of data external to the workspace.

A primary design goal in most of the APL file systems was shared access to data; they not only solved the problem of access to large amounts of data, but also introduced the capability of interaction between different workspaces, allowing the solution of much larger and more complex data processing problems. These APL file systems also gave APL users their first access to the rest of the data processing world. Interfaces were provided that allowed batch programs in the general operating system environment to read and write APL files. APL applications could now take in data from other systems, could provide data to other systems, and in general had some access to devices like highspeed printers and tape drives. It is interesting that this first step in dealing with the rest of the world was to let them access APL files, rather than APL users learning how to access theirs.

All of these APL file systems could be characterized as being designed and implemented specifically for APL. They all tried to emphasize simplicity, elegance, and a fitness for APL. It can be argued to what extent they achieved these goals, but they do achieve them better than file systems designed for other languages and other purposes.

APLSV Announcement

At the 1973 APL V conference in Toronto, IBM announced a new APL product called APLSV. Like its predecessors, APLSV was a complete APL system. The SV stood for shared variables, and the announcement was a major milestone in the development and evolution of APL.

A shared variable is simply a variable that is accessible, under simple access rules, by two asynchronous tasks. Shared variables are not a file system. They are not a message processing system. They are not any kind of a system. They simply provide the basic tool for building an interface between two systems. They can, for example, be used to build an interface to a file system or to a message processing system.

For such a simple and powerful concept, it has taken a long time for them to be understood and used effectively by the APL community. I think the major cause of the difficulty lies in the emphasis that was placed on interfacing with non-APL systems. When shared variables were introduced, APL users were an isolated lot, knowing little about the complexity lying in wait outside. If shared variables could have been used productively within the APL environment first, it would have considerably eased the step in using them to interface with non-APL systems. However, applications of shared variables between workspaces, both of which have a terminal and a user, are somewhat limited. This turned attention prematurely to sharing variables with non-APL systems.

A few very productive interfaces were developed, notably TSIO. The reason for TSIO's success is that it provided access to external data in the standard IBM APL system. It is interesting to note that much of the TSIO usage is to provide a poor man's (in terms of performance and function) emulation of the integrated APL file systems provided by the commercial vendors. Given the potential of shared variables, it is remarkable how little has been done.

The experience with shared variables in SHARP APL, first available in late 1978, has been markedly different. We appreciated the significance of shared variables in 1973, but also saw their limited applicability in our environment at the time. We had a good integrated APL file system and did not need TSIO. Batch access to APL files provided adequate interface with external media and devices. We ran only APL, and did not have any non-APL systems to share variables with. Moreover, we had no intention of getting any as we were dedicated to APL and had no thoughts of diversifying into COBOL or FORTRAN packages.

The use we did see for shared variables was in facilitating communication between APL workspaces. We already did this to some extent with shared files, and saw how nicely shared variables would simplify such applications. However, we also saw how the usefulness was severely limited by the constraint that an active workspace be tied to a terminal and a user. This led to the introduction of N-tasks in early 1976. An N-task (no terminal task) was simply an APL workspace that was put into execution by another APL workspace and did not require a terminal or a user. When shared variables became available in SHARP APL their acceptance was immediate and widespread. I think it is safe to say that in the course of a year the design of large systems was revolutionized by the synergistic combination of shared variables, N-tasks, and the integrated APL file system. Now that we are familiar with shared variables in the APL environment, we can better use them to interface with non-APL systems.

The Corporate Data Centre

In the early seventies, most corporations using APL did so on an outside service bureau. In the course of the decade, APL's success resulted in more and more corporations running APL in their own data centres. Although economic or security concerns were usually used to justify running APL in-house, it really is a much simpler matter of putting the processing power of APL where the data is. More and more data centres run APL as part of a larger data processing complex. Bringing APL time-sharing, which had developed along a unique path for a long time, into what was still fundamentally a batch (MVS) or system programmer (VM) environment has created considerable confusion. APL, as a small part of a much larger and more complex system, entered a new critical phase in its development. As the new kid on the block, there was considerable pressure to conform.

VS APL

IBM bowed to this pressure on APL to conform to the corporate data centre environment with the introduction of the VS APL product. The language remained intact, but the environment was sacrificed and abandoned. VS APL is a language processor whose environment is determined by the host system in which it runs. This results in significant differences, that require considerable user knowledge and awareness of various host environments, none of which are particularly well suited to APL. To make matters worse, the two facilities most critical to serious commercial use of APL, shared files and shared variables, are the areas where the differences are most severe. Moreover, in abandoning control of the APL environment, VS APL, to a large degree, lost the ability to be a high performance system. VS APL, by using the scheduling and resource management services of its various hosts (services designed for other, more general purposes) can not be as effective, or as efficient, as a system which provides its own, specially designed services.

VS APL did make it very easy to get the APL language into the corporate data centre. But it did only half the job, putting in the language, but leaving out the system.

SHARP APL Program Product

Around 1976 it became clear to us at SHARP that although the APL time-sharing service bureau business would continue to prosper, it was in some respects very limited. The potential growth in APL processing easily exceeds the capability of any single data centre. Many of our customers had data centres considerably larger than our own, and they were just filled with data. It is a lot easier to bring processing power to data, than the other way around. The challenge of providing orders of magnitude growth in APL processing could only be met by installing APL systems in the corporate data centres. So it was, that we started the development of SHARP APL as a distributed program product.

Four goals guided the development of the product.

The first was that the product was to be a complete system. Even as a small part of a large data centre, we believed that most applications would best be designed, implemented, and run entirely within the APL system. To do this, a complete system would be required that, in addition to the language processor, also provided facilities like an integrated APL file system, N-tasks, shared variables, and event trapping. If we were to be the new kid on the block, we wanted to be as strong and independent as possible.

The second goal was that the APL system was to be invariant, regardless of the host operating system. Moreover, we wanted the product we distributed to be identical to that which we ran ourselves. In fact, the first releases of our product were exact copies of our production system and, like it, required essentially a dedicated, stand-alone machine. As the system was further developed, we achieved invariance by identifying the host services required by APL, and isolating them in host-dependent interface modules. SHARP APL is the same to the user, regardless of the host. Several of our MVT and DOS installations have, or will shortly have, switched to an MVS host with no conversion or user education. In fact, some night in the coming months, we expect to switch our own DOS based time-sharing service (supporting more than 300 simultaneous users on an AMDAHL V8) to MVS without our APL users noticing.

The third goal was to preserve, and improve on the performance aspects of the system. The key realization here is that the CPU efficiency of the language processor is not the most important aspect of a high performance time-sharing system. In a large system (40 users up to several hundred) the performance of the schedulers, resource managers, and the file system become overwhelmingly important. The design of SHARP APL, and the fact that it is a complete system, gives us control over all aspects of the system. The same high performance schedulers, resource managers, and file system run in all host environments.

The fourth goal was a high performance, shared variable processor. This was necessary just within the APL system, but had substantially increased significance in that it provided the means of interfacing between the APL system and the rest of the environment. The product includes several auxiliary processors that are functionally identical in all host environments. These include an interface giving non-APL systems the same access to APL files as APL systems have, utilities for moving data between APL files and OS data sets, access to highspeed printers, and access to standard SORT/MERGE utilities. A recent development of particular significance is a shared variable interface directly to the APL interpreter. An APL task that communicates through a shared variable instead of a terminal is called an S-task. This is described in the paper **SHARP APL Multiprocessing and Shared Variables** in these same proceedings. The MVS environment also has interfaces to the JES internal reader and writer, and a VTAM application program that supports the IBM 3270 display stations both as APL terminals, and in full screen mode.

The overall view is that it is possible for a complete APL system to effectively co-exist with other systems in a large, complex, operating system environment. Shared variables allow well defined, clean interfaces between APL and other systems. Integration does not imply losing anything, it means gaining a great deal. We have achieved co-existence and have made significant steps towards effective interfaces with other systems.

Futures

At this time, SHARP APL is running as a distributed program product at eleven installations with five of them in MVS environments. Our emphasis on large, production APL systems, interfacing with large data bases and other systems, will mean that the majority of our new installations will be in large data centres running MVS. There will continue to be VM installations, but fewer, as the emphasis there tends to be on development and isolation, whereas the MVS emphasis is on production and integration. We also expect to have installations in the DOS/VS environment.

The traditional strength of IBM time-sharing products in VM, and their relative weakness in MVS have led to a suspicion of high performance time-sharing claims for the MVS environment. We are proving that our approach to the problem will give the user a high performance, complete, APL time-sharing system, properly integrated with shared variable interfaces in the MVS environment. The identical product being available in the other operating system environments considerably eases long range planning and migration concerns.

A major development area will be in providing new and enhanced auxiliary processors. Projects currently under study or active development include TSO and TCAM interfaces, a TSIO compatible product, a general interface to VSAM files, and IMS interfaces at the IRSS (intelligent remote station support), BMP (batch message processor) and DL 1 levels.

The fact that our shared variable processor is upwardly compatible from the ones for the IBM APLSV, VSPC and CICS environments means that auxiliary processors for those systems could easily be made to work with SHARP APL. The next few years will see the development by software houses of a wide range of auxiliary processors.

We expect many new large systems to be developed as hybrid systems, particularly in the MVS environment. The ease with which APL systems can call Batch systems (e.g., the JES interface), and the ease with batch programs can call APL systems (e.g., the S-task interface), will make hybrid systems the preferred solution for many large, complex applications. Increasingly, typically non-APL applications will have some, and then more APL parts.

The fact that many of the installations will be in large data centres with multiple computers gives priority to a very interesting development project, namely, to extend the current concept and implementation of the shared variable processor to provide clean efficient interfaces between tasks, regardless of the particular machine executing them.

The emphasis in this paper is on shared variables and interfaces to non-APL systems. This should not obscure the fact that our real commitment is to APL itself. Eventually, of course, there will be no non-APL systems. To that end, the majority of our development work continues to be in the language and the system itself. Examples of recent developments are extensions to the primitives

compression, and, or, and grade, and the work on operators and enclosed arrays reported by Bernecky and K.E. Iverson [5].

Summary

The APL language and the APL system developed in relative isolation from hardware concerns and from other languages and operating systems. SHARP APL, in direct evolution from venerable ancestors, has became a widely used, high performance, high function, time-sharing system. This system can now, without sacrificing performance or function, be smoothly integrated into the corporate data centre in the MVS, DOS/VS, or VM environments.

The shared variable processor allows effective and efficient interfaces to be built between APL and any other system in the total environment. Several important interfaces already exist.

In the future, no data centre will be without an APL system. If the APL system is high-performance, high function, and properly interfaced, its use will grow very rapidly and it will subsume more and more application areas.

If a proper APL system is provided in the corporate data centre, it will grow and flourish.

References

[1] Falkoff, A.D., and K.E. Iverson, **The Design of APL**, IBM Journal of Research and Development, Vol. 16, No. 4, July 1973.

[2] Falkoff, A.D., and K.E. Iverson, **The Evolution of APL**, SIGPLAN Notices 13 8, August 1978.

[3] McDonnell, E.E., **A Socio-technical History of APL**, APL Quote Quad, Vol. 10, No. 2, December 1979.

[4] Lathwell, R.H., **SHARP APL Multiprocessing and Shared Variables**, in this volume.

[5] Bernecky, R., and K.E. Iverson, **Operators and Enclosed Arrays**, in this volume.

THE APL USER COMMUNITY - ITS ROOTS

Garth H. Foster
Dept. of Electrical and Computer Engineering
Syracuse University
Syracuse, New York

Abstract

The history of the development of the APL user community is traced, with attention given to the formation of the SIGPLAN Technical Committee on APL (STAPL), from 1970 to 1973, as well as its activity since. The role that users of APL have and should play in the development and use of the language are discussed briefly.

Introduction

At some point in the development of a nation, society, or product, enough time has gone by to offer perspective, and the formal writing of a history begins. Sometimes, it is hard to tell whether the effort is intended to be a factual rendering by a healthy organism, recounting how the "good guys" won against difficult odds, or whether the account is that of an entity in the September of its years setting the record straight, before passing from the scene. Whatever the reason, APL is beginning to accumulate a written history, rather than an oral tradition. We no doubt hope that the record is being set down for the growing legions to come, rather than documenting a curiosity for computer history.

Falkoff and Iverson's paper (Falkoff-Iverson 1973) on the design of APL was the first formal accounting of the design effort of APL beyond the terse acknowledgements that had been made in IBM's APL Users Manuals. A more recent paper by the same authors (Falkoff-Iverson 1978) was delivered at the ACM SIGPLAN History of Programming Languages (HOPL) Conference in June, 1978. A video tape prepared as a part of the 1974 Conference in Anaheim (Coast Community College 1974), and a recent paper by Gene McDonnell (McDonnell 1979) are somewhat more anecdotal in nature. These give a good view of the forces and personalities that shaped APL, particularly within IBM.

This paper addresses the interests and activities that the users of APL have had since APL\360 became publicly available. This view of the history of APL activity by the present author is a personal one, and as a result, it may suffer from that perspective.

It is convenient to take 1973 and the events during that year as a watershed, both from a technical point of view of APL as well as that of user activity. First of all, 1973 divides the period from the introduction of APL\360 to the present, roughly in half. When we examine the paper on the development of APL which was written in 1973 (Falkoff-Iverson 1973), and then ask what would have to be added in mid-1980 to update the technical history of APL, from an IBM point of view, we probably come to the conclusion that little new material needs to be added. In fact, when examining APL implementations, one notes that since 1973 and the advent of APL.SV, most of the activity appears to be that of adding capabilities by adding system (quad) names to expand the interface between the user and the system.

The year 1973 is also a dividing point because, until that time, the users were searching for a structure in which to organize their activities. In 1972, The Association for Computing Machinery's (ACM's) Special Interest Group on Programming Languages (SIGPLAN) changed their bylaws to permit the creation of Technical Committees, and when the SIGPLAN Technical-Committee on APL (STAPL) was established in 1973, the first (and currently only) technical committee of SIGPLAN began to provide for user activities in APL.

User Activity Before 1973

After APL was released as a Type III, (contributed) program in the SHARE library in 1968, it was only natural that an interest area for APL would be set up under SHARE's Interactive Systems Project, and an APL Committee was formed. Obviously, a means of communication was needed when the committee was organized at the winter meeting of SHARE in early 1969, and the author offered to edit and distribute a modest newsletter called, SHARE*APL\360. Although the hope was that SHARE would be raised to the power of APL\360, the some 35 copies of issue one did not seem to indicate a large power.

By the time the second issue was out, the first APL user's conference sponsored by the State University of New York at Binghamton (subtitled: **The March on Armonk**) had been held. There were four aspects related to that conference that should be mentioned.

The first is that it was well attended, even though it's subject matter was breaking new ground. APL\360 had been released as a program product about that time, and a number of IBM sales people attended the conference to find out about this new language that they were to sell. Some observers reflecting back over the years, have felt that either the sales team did not learn enough about APL in order to sell it, or else Armonk never knew that a March was taking place.

The second aspect of that conference was that it was clear that interest in APL and the Newsletter was broader than the definition of machines that met qualifications for SHARE membership. An interim step was to distribute the SHARE*APL\360 Newsletter to all those interested. It was felt that such a step would eventually be disliked by the users of the IBM 1130 or 1500, as well as the users of APL systems being implemented on non-IBM machines. Furthermore, SHARE probably would not be pleased because that organization was trying to come to grips with its relationship with IBM, in the world of unbundled software. What should be done was not yet clear, but before the quarterly newsletter was a year old, I renamed the publication the **APL QUOTE QUAD** and made it available to SHARE, and users of non-SHARE, and non-IBM, APL alike.

The third aspect of the Binghamton Conference, due to contacts that I made at or after the meeting, was that a number of APL implementations were under way, and in some cases, uniformity among them was only incidental. In an attempt to foster a greater concern for common interests and problems, I invited representatives of most known implementations to gather at Syracuse University's Minnowbrook Conference Center in the fall of 1969, to discuss differences and similarities among APL systems. Attendance was by invitation only, and this workshop set the pattern for additional visits to Minnowbrook in 1973, 1977, and 1980, as well as the editions of the same series of workshops which were hosted by Queen's University of Kingston, Ontario, in 1976 (at Kingston) and 1979 (at the Asilomar Conference Center in Pacific Grove, California). For the moment, I shall defer commenting on what I think these workshops have accomplished.

The final aspect of that first user's meeting was that it was received with a zeal bordering that of a group of faithful witnessing to the true belief. A lack of objectivity would persist for some time, but the notion of another gathering was easily accepted.

The next meeting was organized in 1970, at the Goddard Space Flight Center at Greenbelt, Maryland. This meeting and the year 1970 saw the first direction toward formalizing an APL community. Early in the year, an APL project had been organized within SHARE as an outgrowth of the APL committee mentioned earlier. The Goddard meeting was not influenced by the presence of a strong APL activity related to IBM, because a number of those present decided to petition the ACM to form a Special Interest Group (SIG) for APL. The required number of signatures were collected and the author was asked to transmit the request to ACM on behalf of the organizing committee. The word that came back from ACM was that it was not appropriate to have a SIG devoted to a single programming language, when there was already a special group devoted to all programming languages (SIGPLAN). In a positive response, the editor of **SIGPLAN Notices** offered to reprint the **APL QUOTE-QUAD**, and this was done beginning with Volume 2, number 2, of the **QUOTE-QUAD**, with its distribution continuing as before. An ad hoc meeting was held by the steering committee at Lakehead University, Thunder Bay, Ontario, where A. McEwan and D. Watson, Editors of the **APL QUOTE-QUAD** were employed. It was felt that continued publication of the **QUOTE-QUAD** should be of first priority and that either there should be a discontinuance of seeking a user community crossing vendor lines or else a separate organization should be formed. These choices would be put before the next meeting, which was to be held at the University of California at Berkeley, in 1971. Continued effort by those editors and L. Gibson, M. Pritchard, and A. Anger, who were to follow, saw that the first priority was met.

The Goddard and Berkeley meetings in 1970 and 1971 were similar, in that they were more like workshops than conferences. There was no call for papers and, of course, no formal proceedings were published, although summaries of the meetings were recorded in the **QUOTE-QUAD**. The meetings were organized in an informal manner. Plans for a next meeting were undertaken as soon as a host could be found, and this usually meant that the next year's meeting was planned at or after the current one, a practice which cannot be tolerated now-a-days.

When the options for a user organization were presented at the Berkeley gathering, the consensus was that the notion for an independent user community for APL should not be abandoned, and when those ready to propose a new organization were about to make a motion to that effect, A. Perlis suggested that perhaps ACM could be persuaded to reconsider. He and others undertook that task, but until some formal arrangement could be made, one way or another, the ad hoc user activity would continue.

In the short term, user activity in North America was influenced by the first international conference in Paris, in 1971. That conference was organized independently from the meetings being held in the United States, and the organizers of Colloque APL had both invited and contributed papers and issued a proceedings. Clearly, there were two lessons to be learned for the activity based on this side of the Atlantic: the APL user community was international and proceedings gave 'a longer term value to the meeting.

Thus, the meeting of 1972 in Atlanta, Georgia, had a call for papers, issued a proceedings, and was numbered as the fourth in the series. Unfortunately, Roman numerals, after the original mode of enumerating the SHARE meetings, were used. Mention is made of this to emphasize that the development of APL user activity was as much by accident and misstep, as it was by design.

As mentioned earlier, SIGPLAN did change its bylaws to permit the formation of Technical Committees (TECS) in 1972, and application was made to ACM and SIGPLAN to form STAPL in that year. It seemed that a vendor-independent users community was going to come about after all.

Volume 3, issue 4 of the **QUOTE-QUAD** turned out to be the last issue to be published as part of **SIGPLAN Notices**. This decision was not related to the application to establish STAPL, but rather because when the **QUOTE-QUAD** came out in **SIGPLAN Notices** the APL material was larger than the rest of the issue. SIGPLAN's publishing budget was being used to reprint the **APL QUOTE-QUAD**.

In 1973, APL V was held in Toronto, marking the first time that one of the series of meetings started in 1969 was held outside the United States. There was also a separately planned meeting in Copenhagen, APL Congress 73. It was decided that having two meetings that year, with a third almost being held, was too many. An agreement was reached, whereby meetings would alternate between North America and Europe, with odd numbered years held in Europe.

The STAPL application was accepted in 1973, although it took another year to get the financial matters straightened out, and have the membership roll of the ad hoc group transferred to the ACM. This process was further complicated and delayed because, at the time, the ACM was changing over to have their data processing done by the Institute for Electrical and Electronic Engineers. The additional complications meant that the creation of STAPL was stretched out into 1974, although the structure came into being in 1973. Officers were appointed for 1973-1975, with the present author serving as Chairman. Since the administrative structure was still in formation, plans were made in 1973 to have the 1974 Conference sponsored by the Coast Community College District of Costa Mesa, California, but to hold STAPL responsible for loss or profit. The **APL QUOTE-QUAD** was passed on to STAPL as a membership benefit.

In the end, STAPL came about because of the work of a large number of people and fortunate circumstances. People and organizations made conferences and workshops come about, even when there was no source of start-up funds and no way to absorb losses if attendance turned out to be poor. Interest in APL had kept growth at a slow, but steady pace.

APL User Activity After 1973

With the announcement of APL.SV at APL V in Toronto, IBM's APL had access to the world outside APL, including file systems through shared variables. This point can also be taken as the time from which APL had a greater impact on business and active growth, in terms of the size of conferences that took place. The size of the crowd at APL 6 at Anaheim, in 1974, was larger than anticipated and STAPL was financially healthy from inception.

The 1975 conference in Pisa, Italy, saw STAPL as a co-sponsor, with most of the editing of the conference proceedings being done in the United States, with printing done by STAPL through the ACM. While the logistics of having an international program committee, a strong local arrangement committee, and producing the conference and proceedings on opposite sides of the Atlantic are difficult, this conference showed that such cooperation is feasible. The numbering scheme of: APL nn, where nn gives the year, was adopted for 1975 and intended for subsequent STAPL sponsored user conferences.

The first STAPL elections were held in 1975, with the author elected as chairman. The officers and executive board made several attempts to find a host site for a 1977 conference in Europe, but none was found. By that time it was recognized that a successful international conference was going to require approximately eighteen months in planning and execution, and it became clear there would be no conference in 1977, following the pattern of reserving odd numbered years for European conferences.

Ottawa was the site for APL 76 and from organizational and attendance points of view, it was more successful than previous meetings. Meanwhile, the officers of STAPL were torn between the views that: (1) STAPL should hold conferences less frequently than every year, so that the quality could be improved, or (2) STAPL should cease holding conferences altogether, in favour of publication of papers at other conferences and in traditional journals of computer literature. By the time the lack of a 1977 conference was felt, it was too late to develop plans for APL 78. Newly elected officers of STAPL under the leadership of Phil Abrams sought the leadership that would produce an APL 79, with the intent of having a STAPL conference every other year.

The absence of an APL 78 saw the emergence of the first I.P. Sharp Users Meeting as a commercially sponsored vehicle for telling the APL story in general, and that of a successful service and program product in particular.

The success of APL 79 in Rochester, New York, with almost a thousand persons in attendance indicated that interest in APL had not waned with the absence of STAPL sponsored conference in 1977 and 1978, or with the presence of a 1978 APL meeting devoted to a particular vendor. Planning was undertaken in 1979 for APL 81 in San Francisco, and the change of officers in that year placed the reins of leadership in the able hands of Gene McDonnell. The current officers have been able to make headway in furthering the interests of STAPL within ACM, that previous sets of officers could not achieve.

As this material is being written, it appears that the necessary steps to transform the technical committee on APL under SIGPLAN, into its own special interest group are underway. When accomplished, this will achieve a goal that was first sought ten years ago at the Goddard meeting.

The APL user community is alive in STAPL, but not without problems.

This narrative has intended to set down a personal view of the development of user activity. As a result, there has been more emphasis on the historical developments that brought us to this point, and less on the value of what has (and has not) been achieved along the way.

Some Observations on What Has Been Accomplished

STAPL, a strong, financially stable, user organization has been established in a recognized, professional society, the ACM. Because it has members all over the world, the ACM is international in scope. A central office to maintain membership records and to provide a single place for ordering publications has been obtained without creating and staffing it.

The publication of an independent source of APL news, views, theory, and practice continues, although the publication of the **APL QUOTE-QUAD** by volunteer efforts, has been and will continue to be a concern.

While the conferences and their papers, presentations, exhibits, and proceedings were useful in their own right, a number of conferences were occasions for extending the scope of APL. It is interesting to note that APL.SV and APL/CMS were introduced at the time of APL V and APL 6 respectively, rather than at an IBM gathering, such as a SHARE or GUIDE meeting. The publication of the IBM international standard, as an appendix to Falkoff and Orth's paper at APL 79 provided an important document which is being used as a base document for the **International Standards Organization** (ISO) working draft and the **American National Standards Institute** (ANSI) APL project X3J10.

The Implementer's Workshops have offered opportunities to discuss a number of topics for extensions to APL, including complex arithmetic and nested arrays. The ongoing discussions among the implementers of various APL systems have been useful, and the Workspace Interchange of Source (WSIS) is a good example of cooperation across vendor lines.

On the balance, the creation and growth of a vendor-independent APL user community has been a positive force in the overall development of the language. There are some areas where concern for the future of the user and APL interface needs to be expressed.

APL Users and Tomorrow

First, we should not lose sight of the fact that STAPL exists. It is easier to strengthen it than to discard it and form a new activity. STAPL is a volunteer organization and more users need to get involved with it. Expecting the same people to keep STAPL going will eventually result in its demise. If you leave here with the belief that getting sent to a conference to hear, or better yet, deliver a paper is all that is necessary, you have missed the mark. Someone has to see that there is going to be a conference to attend.

Secondly, one should note that 1980 has seen the Practical APL Conference, sponsored by STSC, APL 80, in the Netherlands by the Dutch Computer Society and the European Cooperation in Informatics, as well as the present conference. APL 80, which was planned apart from STAPL, may well have suffered from oversaturation, due to the fact that it was competing with both the STSC and the I.P. Sharp Users Meeting for attendance. Certainly it was not advertised in North America nearly as well as either of the other two conferences this year.

There is certainly room for meetings of Sharp or STSC users in the scheme of the APL community, just as there is a continued need for SHARE, GUIDE, and SEAS projects for IBM products.

User groups of other vendors of APL should be encouraged to organize and promote APL activity. The meetings and proceedings of the STSC and Sharp Conferences promote APL in positive ways that go beyond say, the long standing user activity in SHARE or SEAS.

At the same time, those of us who are users of the I.P. Sharp program product should consider whether a more formalized form of mutual cooperation, study, and influence might not better serve our common interests than our separate attendance of these meetings.

I still strongly support the ideas of a vendor-independent user community, because the fundamental truth is that despite our labour and success in forming the APL user community, the acceptance and use of APL is still not very great. Languages such as PASCAL and ADA have come onto the scene since the introduction of APL. The design, introduction, use, and standardization of these languages have been advanced by forceful user activity, including universities and computer scientists. The users of APL can no longer wait for this marvelous tool to be picked up and refined by the theoreticians. APL's productivity and power must be demonstrated in the marketplace by those who use it.

References

(1) Coast Community College District, Costa Mesa, California, videotaped at the sixth International APL User's Conference, Anaheim, CA, May 14-17, 1974.

(2) McDonnell, E.E., "The Socio-Technical Beginnings of APL", **APL QUOTE-QUAD** 102 (December 1979), pp. 13-18.

This paper was orginally presented in September 1979 in Copenhagen, Denmark at a meeting celebrating ten years of commercial use of APL in Scandinavia.

(3) Falkoff, A.D. and K.E. Iverson, "The Design of APL", **IBM Journal of Research and Development**, 174 (July 1973), pp. 324-334.

(4) Falkoff, A.D. and K.E. Iverson, "The Evolution of APL", **SIGPLAN Notices**, 138 (August 1978), pp. 47-57. (This issue contains preprints the (HOPL) History of Programming Languages Conference, June 1-3, 1978, Los Angeles.)

A Chronology of APL User Activity

Year	Conference/ Workshop	User Activity
1969	First APL Users Conference "The March on Armonk" State University of New York Binghamton, NY, USA	Issue 4 of SHARE*APL\360 Newsletter renamed APL QUOTE-QUAD.
	APL Implementer's Workshop Minnowbrook Conference Center Blue Mountain Lake, NY, USA	
1970	APL Workshop Goddard Space Flight Center Greenbelt, MD, USA	Starting with Vol. 2, No. 2 of APL QUOTE-QUAD published in SIGPLAN Notices. Application made to ACM to form special Interest Groups. Ad hoc committee meets in Thunder Bay, Ontario.
1971	APL Workshop University of California Berkeley, CA, USA	
	Colloque APL IRIA Paris, France	
1972	APL IV Atlanta Public Schools/Georgia Institute of Technology, Atlanta, GA, USA	By-law change in SIGPLAN to allow formation of Technical Committees (TEC's).
		Vol. 3, No. 4 APL last issue to appear in SIGPLAN Notices.
		Application made to SIGPLAN to form STAPL as a TEC.
1973	APL V STAPL Toronto, Canada	
	APL Congress 73 I/S Kommuncdata and NEUCC Copenhagen, Denmark	Application and bylaws for STAPL accepted.
	APL Implementer's Workshop Syracuse University Minnowbrook Conference Center Blue Mountain Lake, NY, USA	

1974	APL 6 Coast Community College District Anaheim, CA, USA	Administrative structure and budget for STAPL established.
1975	APL 75 CNUCE/University of Pisa/STAPL Pisa, Italy	First STAPL election. G. Foster, Chairman
1976	APL 76 ACM-STAPL/Ministry of State for Science and Technology Ottawa, Canada	
	APL Implementer's Conference Queen's University Kingston, Ontario, Canada	
1977	APL Implementer's Workshop Syracuse University Minnowbrook Conference Center Blue Mountain Lake, NY, USA	STAPL election. P. Abrams, Chairman
1978	I.P. Sharp Users Meeting I.P. Sharp Associates Toronto, Canada	
1979	APL 79 STAPL Rochester, NY, USA	STAPL election. E. McDonnell, Chairman
1979	APL Implementer's Workshop Queen's University Asilomar, Pacific Grove, CA, USA	
1980	Practical APL Conference STSC, Inc. Washington, DC, USA	ANSI Standard Project for APL X3J10 is begun.
	APL 80 Dutch Computer Society and European Cooperation in Information Noordwijkerhout, Netherlands	
	APL Implementer's Workshop Syracuse University Minnowbrook Conference Center Blue Mountain Lake, NY, USA	
	2nd I.P. Sharp Users Meeting I.P. Sharp Associates Toronto, Canada	

LEGI-SLATE BILL AND VOTE TRACKING SERVICE

Curtiss C. Grove
Legi-Slate
Dallas, Texas

At the outset, it seems fair to confess that I may be participating in this APL "users" meeting under false pretenses. In the sense that I assume I.P. Sharp **intends** its customers to be "users" of APL, I do not qualify. I don't know a quad from a catenate, and I frankly don't care about such things. But I admit that I do wonder about why APL insists on hiding perfectly good keys — like parentheses and the question mark — in strange places on the keyboard.

A Different Kind of APL User

In a different sense, our company does qualify as a moderately active "user" of APL. LEGI-SLATE is a bill and vote tracking service. The idea for what was to grow into LEGI-SLATE was born in 1975, with a friend on the other end of a telephone call telling me that what I had just suggested as a sensible use of a computer was probably not feasible. From that discouraging beginning, we began a slowly building effort to gauge the market for the envisioned service. As is often the case with a new idea, prospective customers had difficulty in reacting to our oral descriptions of the service: we had to be able to demonstrate it to them. So we began development of a very modest pilot version related to the Texas Legislature. We found enough encouragement to aim toward launching the service in late 1978, in time for the Texas Legislature in 1979.

Why Texas? Because we are a Dallas-based company, and Austin is only 200 miles away. The Texas Legislature was our shakedown cruise. At that time, we had plans to move on to other states that have year-round legislative sessions, instead of the single, five-month session Texas has every two years. But as we showed LEGI-SLATE to prospects around the state, we got one of two reactions:

Either: "Yes, I need that for the Texas Legislature, **but when are you going to Congress**?"

Or: "I'm not that interested in the Texas Legislature, **but when are you going to Congress**?"

It didn't take long for us to decide that California, Illinois, New York and other major states would have to wait while we looked closely at the potential for LEGI-SLATE in Congress. After several months of preparation, we made the commitment to "go to Congress". We opened an office in Washington in September of 1979, and began catching up with the past activities of the 96th Congress, then midway in the First Session. By year end, we were caught up and ready for daily updates of Congressional activity, beginning with the Second Session, which started in January of 1980. Marketing in Washington began at the end of last January.

What is LEGI-SLATE?

That's the history, but what **is** LEGI-SLATE? It is a computerized bill and vote tracking system designed for people who know nothing about computers, but need to know quickly about Congress — or the Texas Legislature, or any other legislature.

Each day, we enter a synopsis of every bill and resolution introduced, then track all actions that take place on those measures. Part of our effort includes reading every measure at least twice to assign keyword codes that describe the contents of the measures. This then allows our users to search all measures introduced, to find those on subjects of special interest. And during the course of a typical two-year Congress, between 15,000 and 20,000 bills and resolutions are introduced by the members.

An important feature of LEGI-SLATE is that we also enter all recorded votes, by names of individual members. This leads to a variety of ways our users can analyze votes of particular interest among the 435 House members or 100 Senators.

The KISS Principle

As we say in our slogan, "LEGI-SLATE puts Congress at your fingertips", but only if you have a computer terminal and your fingertips can find the right parenthesis and the colon on the keyboard. As already emphasized, LEGI-SLATE is designed for people who know nothing about computers, and the majority of them do not. Thus, we work hard to live by the KISS Principle: "Keep It Simple, Stupid!".

I lost one sale because the executive I was dealing with simply could not get the hang of holding the shift key down to make a right parenthesis. And in another demonstration, a different executive really could not "find the 'a' on the typewriter". For the most part, our customers do **not** use terminals with the APL character set. Therefore, we restrict anything **they** see in LEGI-SLATE to characters that are universally available on any terminal they might use.

We hold our customers in something called "program control", which means that they cannot escape from LEGI-SLATE to do other things on I.P. Sharp's computer (at least not through their LEGI-SLATE account number). But at the same time, we have developed what I now hear programmers refer to as a very "friendly" or "forgiving" system. If our user begins wandering down the wrong pathway, LEGI-SLATE will take you gently by the hand and lead you back to where you should be — **all in plain English**. We even discourage use of the backspace (or — heaven help us — the "CONTROL H"), because then you must also remember to "LINE FEED". Otherwise, you will get something called a "CHAR ERROR", which I have come to learn is computerese by someone who was unable or too lazy to spell "CHARACTER ERROR". But even spelled out, the problem thus created by the innocent user is not well defined, much less its solution. So instead of backspacing to make a correction, we instruct our users simply to press RETURN and try again.

We have worked very hard to make LEGI-SLATE simple and obvious to use. At times, old-pro programmers are scornful of our simplistic appearance — until they have used it for a time. Then they get hooked and complimentary. Even so, we find surprises.

We firmly believe that the whole world should be confined to 8½ by 11 paper, vertical, 80 columns wide, and preferably with a three-hole punch on the left side of the paper. Thus, if you are ready to print a LEGI-SLATE report, it would be nice to have it begin just **after** the perforation, across the page, not an inch or two **above** the perforation. So we provide an option that lets you pause to "POSITION PAPER" before you start printing. But in the context of **our** users, we found people asking, "You mean I'm going to get a "position paper?" So we changed it to read "RE-POSITION PAPER".

Who Uses LEGI-SLATE?

Our subscribers include corporations, trade and professional associations, labor unions, universities, law firms, savings and loans, chambers of commerce, political action committees, political parties, the media, and government agencies. One of our more pleasant discoveries was how anxious federal agencies are to know about what Congress is doing.

We have come to learn that the interests and needs of the local lobbyist on the scene are much more narrowly focused than in his organization's office back home. And while some customers are mainly interested in analyses of the members' voting behavior, others have no interest in voting, but only the progress of certain bills through Congress.

LEGI-SLATE depends heavily on personal selling and on-site training. Shortly after LEGI-SLATE was launched in Washington, we got numerous calls from most of the familiar timesharing, data base companies who wanted to convince us why LEGI-SLATE would be better off in **their** computer and their catalogue of data bases. We politely discouraged most of them by asking whether they provided APL. You see, even though I'm not an APL **user**, I have learned to be an APL **loyalist**. We are absolutely convinced that we could not have developed LEGI-SLATE in all of its simple-appearing complexities as quickly as we did in any other language. The few other companies that do offer APL were gently put off by our expressions of respect and admiration for I.P. Sharp — both as a person and as a company.

As we got underway in Washington, we learned of LEGIS, the internal bill tracking system operated by the U.S. House of Representatives. There had been some talk of making this system available to the public through a timesharing vendor, but that effort never got off the ground. And even in their internal system, rank and file members are not allowed access to voting records in the Congressional computer — only the leadership of each party, respectively, can use the computer to see how their fellow members voted. Development of LEGIS began in 1969, and they went on the air in 1977 — eight years later. And they have estimated that it would take them about two years to program the analytical tricks LEGI-SLATE already offers for voting records.

Daily Operations

Operationally, here is how LEGI-SLATE works. The basic source is the Congressional Record which, miraculously, is published every morning of working days after any day when the House or Senate was in session. Our office is located near the Government Printing Office, and we get six copies delivered to us about 15 minutes after it is first available.

Several related tasks are assigned to our editors. Some begin coding recorded votes.

Others begin a coded diary of all actions that took place on the House and Senate floor. Others code members who have co-sponsored bills or resolutions. Others code committee actions of the previous day, and others begin reading bills to assign keyword references.

While this coding is still underway, data entry operators begin entering a synopsis of each bill and resolution that was introduced the previous day, then as coded entries are ready, the data entry operators enter votes, cosponsors, floor and committee actions, and so forth. After all entries have been made, proof lists are dumped back out of the data entry terminals for proofing — by different editors and always against the original source, not the coded input forms. After errors have been edited, the files are updated and immediately available to LEGI-SLATE subscribers. From start to finish, this daily routine lasts from about 9:30 to noon — sometimes, 1:00 or 2:00 in the afternoon if the Congressional Record was late or unusually large.

While such an operation might be possible in a batch operation, I would hate to be responsible for it! Clearly, one of the great appeals of our use of APL and our association with I.P. Sharp is the speed, simplicity and reliability with which we can enter, edit, and update the data we provide to our subscribers.

Down With "Bit Wizards!"

But I would be remiss if I didn't comment about those occasional announcements such as, "LINES DOWN" or "APL DEFINITELY HUNG" or my favorite of all, "AWAITING FUNCTION FOUR". In the handbook we provide for our users, we quote these mysterious terms they may see from time to time, and then explain that all they really mean is, "SOMETHING BROKE. GO HAVE A CUP OF COFFEE, AND TRY AGAIN IN A FEW MINUTES".

In fact, we go even farther to protect our users from computerese when one of our own programs crashes. If you are a user of APL in the normal sense, you are familiar with alarms such as: SYNTAX ERROR, FILE ACCESS ERROR, VALUE ERROR, FILE TIE ERROR, and all the rest, then followed by a line of gibberish that I have learned is a clue to the programmer about what went wrong. Apart from confusing our customers, that could also let them escape from our program control, free to cause all sorts of mischief. So instead, we let them see only this message:

```
OOPS! WE HAVE A PROBLEM -- PLEASE TRY SOMETHING
DIFFERENT.
```

If they persist and get three strikes (OOPS!), they're out, and we bump them off the system with this message:

```
TOO MANY PROBLEMS ENCOUNTERED. PLEASE PHONE LEGI-
SLATE FOR ASSISTANCE:
202/547-4140 OR 214/630-3462 OR 512/474-8881.
```

The really nice part of this is that **our** systems wizard back in the office can detect when a subscriber has gotten an OOPS!, and **he** gets the gibberish that describes what went wrong, not our subscriber. We frequently surprise a customer when we call later to explain why they had the problem and that it won't happen again.

With this kind of handholding, we try hard to eliminate much of the mystery surrounding computers for our typical user — and we work hard to protect them from "bit wizards", my name for programmers who can't resist the one-upmanship that feeds their programming egos.

LEGI-SLATE was designed for and is marketed to **end users**. This sometimes offends librarians and the new breed of "Information Managers" who prefer to hide the keys to the data base somewhere in their card catalogues. At a seminar on data bases in Washington last spring, one speaker discussed "Selective Dissemination of Information" and showed a diagram of how end users must depend on Information Managers for access to data base information. Not with LEGI-SLATE. One of our earliest and most enthusiastic subscribers is an executive who has a CRT dedicated to LEGI-SLATE on the credenza behind his desk. He did not want a printing terminal, because he would then "have to pay someone to file the paper". And besides, he asked, "Can't I use LEGI-SLATE's computer as my filing cabinet?" We assured him that he can, but he is the exception. Few other LEGI-SLATE customers have moved that far into the "office of the future".

Future Plans for LEGI-SLATE

When we began LEGI-SLATE in Texas, people asked, "When are you going to Congress?" Now that we are in Congress, they ask, "When will you have **all fifty** states?" And Ian Sharp, always on the alert for the Commonwealth, has asked why LEGI-SLATE won't work in the Canadian and British Parliaments. Based on some early investigation, it will.

Then there is REG-ULATE, the companion service for the Federal Register and the regulatory side of the world. We found this to be a natural follow-on product after LEGI-SLATE, with marketing scheduled to begin in the third quarter of 1980. We know this to be a much larger and broader market than that for LEGI-SLATE, requiring an expanded and more diversified marketing program.

My skeptical friend on the other end of that fateful telephone call in 1975 is now persuaded that LEGI-SLATE is real and here to stay. Also persuaded are the firms who now approach us and say, "Gee! **We** were going to do that ... ".

FINAPL - THE DEVELOPMENT OF A FINANCIAL DIALECT OF APL

Geoff Kemmish
Fars Computer Services Ltd.
Manchester, England

Introduction

This paper discusses the environment within which a financial modelling dialect of APL was seen as a useful development, the form such a dialect eventually took, and experience with it at a number of user sites.

The development of FINAPL began when my requirements for financial modelling facilities outgrew those available from more conventional time-sharing packages, and after some experimentation it became clear that an APL-based financial modelling "language" could be developed which would offer better facilities to the user, but at the same time be smaller and require less maintenance.

It should be emphasised at the start that the experience leading to the writing of FINAPL and gained in developing and using it, has been within United Kingdom based companies. It is possible that experience elsewhere might invalidate some or many of our assumptions, and render the basic concept questionable.

Financial Modelling

It is perhaps appropriate to discuss in some detail the concept of financial modelling under consideration here.

The term has been defined in many different ways. All too often these definitions have created an atmosphere of pseudo-sophistication, more appropriate perhaps to the academic world than the world of business. In essence, it is a very simple task — the process of developing financial statements and ratios dealing, typically, with future projections based upon clear, simple assumptions and logical relationships.

When an accountant takes a set of assumptions and historical ratios and applies normal accounting logic in preparing and summarising the spread sheets he is using, the end results are called financial projections or pro-forma statements. Following the same process but using a computer rather than a pencil, paper and a pocket calculator, is financial modelling.

A financial model is a mechanised way of preparing financial projections. The advantages are basically three-fold:

i) the time required to prepare and revise the projections
ii) the cost of preparation

iii) the accuracy of calculation

Financial modelling must be viewed as a functional tool rather than as a technical achievement. How and where can it contribute to reaching corporate goals?

The answers to that question have gone through three phases, characterised sometimes as "bottom-up", "top-down" and "inside out".

"Bottom-Up"

The bottom-up approach to modelling arose from "first generation" use of computers in business. Management was presented with a powerful tool which not only promised to increase the speed and reduce the cost of much clerical work, but also held the prospect of attempting operations that had never been possible previously.

Early successes at improving routine tasks, like payroll and ledgers whetted the appetite of management for new uses of computer technology. However, with management information systems and data bases still in the future, the manager was dependent on the advice of computer experts to guide him to the new applications he sought. Since these experts were often trained in mathematical or scientific disciplines and seldom had experience of general management problems, there was a tendency for new developments to reflect a narrowness of training and experience.

One direction of development was the development of models of aspects of the business and the analysis of the behaviour of these models under various assumptions. Usually the parts of the company chosen were operating units — a plant, distribution centre, or even a single department. Once such an operating model had been demonstrated to senior management, it was assumed that such a model could be expanded to help in the planning process. The reasoning is impeccable:

i) If a model can be developed to cover one function, such as distribution scheduling at one plant, then models can be built to cover other functions at that site.

ii) If models can be developed for all the functions at one site, they can be developed for all the company's sites.

iii) If all these models are put together and a "corporate functions" model added, the result is a model of the whole company's sites.

The reasoning may be impeccable, but the results are disastrous. Attempts to build models in this way, from the bottom of the business upward, led to unwieldy monsters that, had they run, could have done a great deal of harm. Fortunately, most collapsed under their own weight and complexity.

Two lessons emerged from this phase:

i) Operating models are not planning models — their information requirements, processing and results are radically different.

ii) Planning models cannot be built by specialists who are not familiar with the company and its planning processes.

"Top-Down"

The second phase is characterised by an increasing awareness by senior managements that computerised models of their business were feasible, and by a wider availability throughout many organisations of people with experience in using computers to solve problems. Perhaps the greatest influence, however, was the rapid spread of corporate planning, and the perceived need for that planning process to be supported by arithmetic justification. The plans were imposed from the top, and so were their associated models.

Since they were often produced within a corporate modelling unit grafted onto the side of the organisation chart and, all too often, peopled by outsiders, such an activity was naturally resented by existing managers. Given the virtually impossible task of building the necessary models during slack periods in the planning cycle, the result, almost inevitably, was a mechanized version of the existing planning process.

Designing and implementing such a system required that people throughout the company submit to an inquisition about the decisions for which they were responsible, the information on which they based those decisions, how they made those decisions and how they handed them on up (or down) the line. Nothing could be more certain to make a middle manager in a staff function, perhaps already suspicious of the new technology, feel threatened. The result was misinformation, by omission, inaccuracy, or exaggeration. Even those who tried to take the exercise seriously found it difficult to respond sensibly to glib questions such as "How do you make decisions?".

The resulting models looked down from the overall flow of funds, profit and loss and balance sheet into the operations of the business, all too often expressed only in financial terms. They too required large amounts of data and produced, after much expensive calculation — usually on an external time-sharing bureau — vast amounts of output which no one had either the time or inclination to examine carefully.

The lessons from this phase seem to be:

i) Top management commitment to a corporate model is not enough — managers right through the organisation must understand and support what is going on.

ii) All-inclusive models pose formidable design and data problems. Moreover, they acquire the characteristics of conventional, large, DP systems, but with non-DP managers in charge of them.

"Inside-Out"

If, as this experience tends to suggest, company-wide models really are not feasible, where is financial modelling heading?

The answer appears to be anarchy: fragmented — and very probably sub-optimal — models are devised as the need arises. People are increasingly looking for ways of using computing power within their planning tasks. Their needs are often for small, specific models that are, in truth, little more than "super-calculators".

This differs from the previous approaches also when we come to accuracy: such a model exists only to support its manager-user in his work. Questions about the realism of assumptions, the accuracy of data and calculations cease to be relevant. What matters is not whether a model is a correct, or complete representation of reality, but that it does the job.

Finally, there is the question of time-scale. Models built to meet a specific problem-solving need lose their usefulness once the problem is solved. They automatically have a life that is less than that of the problem, for it is not until the problem is recognised both as a problem, and then one for which a model can help, that the process of building the model can begin. Thus a cash-flow problem that lasts some months may eventually give rise to a model with a life of weeks. Something with such a short life cannot justify a great investment in its building.

The need is for tools that can help in "quick and dirty" problem-solving rather than in building lasting systems. However, this approach is not infallible either and there are three guidelines that have to be followed:

i) The manager must be involved in designing, building and using the model.

ii) Simple models are best.

iii) Planning involves the use of models (plural), rather than a single model.

Why APL as a Base?

Given an established realisation that financial modelling is different in kind from ledger-type work, it is obvious that a different range of tools is required: I suspect that very few people would argue that COBOL is a suitable language for modelling applications.

I would go a step further and maintain that, often, modelling requires a different set of tools from those required even in budgeting. However, the distinction is blurred and I doubt whether many would agree with me.

At this stage, however, there is obviously wide agreement on the type of tools required. These can be provided in one of two ways — either via a specific interpreter, written in Assembler, perhaps or PL/1, or by exploiting the APL interpreter. Choosing the second route, which is what we have done in developing FINAPL, opens up other possibilities and raises some difficulties.

Features of "pure" APL that can be of immediate use in modelling include:

i) The highly modular structure of systems written in APL. This is a useful discipline on the model-building process: and also means that, for example, after a full study has been run, parts of it can be run again to examine alterations, and

ii) The array-handling properties of APL as a language mean that often, multiple cases, divisions, or time periods can be handled simply by changing the shape of the data rather than by re-writing program loops.

Properties of "pure" APL that can raise problems, in our experience include:

i) Error handling. Obviously ⎕*TRAP* and ⎕*SIGNAL* have greatly improved the situation, but having decided not to write a customised interpreter, it becomes a matter of picking up the pieces after the user has made a mistake rather than preventing him from doing it in the first place. To be honest, we have not yet got a satisfactory global *⎕TRAP* for FINAPL.

ii) Inconsistencies in dimensions and rank of "small" variables. To an APL-er the differences between nulls, scalars and single-element vectors, and in their effects when used as indexing expressions are — at least — comprehensible, but they can be confusing, to put it mildly!

Why Write FINAPL?

Inevitably, we must ask why it was necessary to write a system such as FINAPL, and to consider developing successor systems, when in a language like APL the sort of facilities that other languages have to be persuaded to provide are immediately available.

The answer lies in the market which we have chosen to attack. There are many — in the U.K. alone, nearly 40 — financial modelling systems available on the non-APL time-sharing market. They all attempt to provide a "language" appropriate to the tasks to be carried out. Most do this by interpreting, interpreting and then compiling, or compiling, "high-level" instructions. The amounts of code involved are, of course, huge. The source code listings of one of the more successful U.K.-based products were six foot high when I last saw them, a few years ago.

Using APL as a base one can develop the kind of vocabulary offered by such systems in a more maintainable form. However, there is a problem of compatibility with the rest of APL. One can choose one of two routes:

i) The market has become used to the idea of variables containing vectors of numbers only, so it would be possible to develop a system that dealt only with vectors.

ii) The alternative, and more challenging, route is to seek to maintain compatibility with the rest of APL, and write the system to behave as APL would were the facility involved an APL primitive. This is the route we took with FINAPL.

One problem that arises, in the second approach, is that often one is almost, but not quite, simulating extended scalar conformability. Some examples might help. *GROW* is an FINAPL function that applies percentage growth rates. Thus, in a five period environment:

```
      100 GROW 10
100 110 121 133.1 146.41

      100 GROW 12 10 8 10
100 112 123.2 133.056 146.3616

      100 110 120 130 140 GROW 10
100 121 145.2 173.03 204.974

      100 110 120 130 140 GROW 12 10 8 10
100 123.2 147.84 172.9728 204.90624
```

```
      100 GROW 2 1ρ10 15
100 110 121.00 133.1000 146.410000
100 115 132.25 152.0875 174.900625

      (2 1ρ100 150) GROW 10
100 110 121.0 133.10 146.410
150 165 181.5 199.65 219.615
```

and so on. In other situations, however, it becomes necessary to produce results that are unique, but reasonably consistent with the APL environment. An example is the FINAPL *NPVAT* function, which calculates the net present values of one or more cash flows at one or more discounting rates:

```
      ¯1000 125 150 150 1100 NPVAT 10
101.6153268

      ¯1000 125 150 150 1100 NPVAT 10 15
101.6153268 ¯50.32679271

      (2 5ρ¯1000 125 150 150 1100 ¯1200 85 105 105 1300) NPVAT 10
101.6153268
¯69.1448672

      (2 5ρ¯1000 125 150 150 1100 ¯1200 85 105 105 1300) NPVAT 10 15
101.6153268  ¯50.3267927
¯69.1448672 ¯234.3734478
```

The second — and major, problem — in designing and implementing a system such as FINAPL is report specification. The *⎕FMT* primitive is an enormous advance, but people using competing FORTRAN-based packages do not specify their reports using WRITE and FORMAT statements. It becomes essential to produce a very-high-level report specification "language" that, for example, handles continuation pages automatically when the default page size of 132 characters by 66 lines is changed to 80 by 20 to fit on a VDU screen. We believe that we have reached the point where the report writer used in FINAPL is comparable with those on offer in the best non-APL systems. Almost inevitably, however, it does not meet the standards of good APL practice exemplified in the discussions on direct definition. I'm afraid we even have niladic functions that return both explicit and implicit results!

Experience to Date

Within the kind of financial modelling activity defined earlier, it is obvious that the type of analysis required at any one time is unpredictable and that any system must offer a very wide range of facilities. Using the approach adopted for FINAPL, we can extend its facilities in almost any direction if asked. More importantly, interfacing to other systems, such as MAGIC, can be carried out without too much difficulty. That extreme flexibility has been a welcome by-product since the original design aim was a compact, easily maintainable package.

We, quite deliberately, left FINAPL as an in-workspace system. APL file systems seem to vary so much that tailoring is the order of the day. Again, where file-handling is available either directly or through a similarly constructed system, there are few problems.

The report writing side has proved to be a great success. Often, we have found that the report specification for a user's system can be left until the end and takes only a very short time to write.

Personally, I have been disappointed that the more modelling-biased facilities have not been taken up with quite the enthusiasm I had hoped, but it was only when the development process was well advanced that I discovered my views on financial modelling, and its place in management were unusual, and that most people felt that facilities such as risk and impact analysis were unnecessary for their tasks.

COMMERCIAL APPLICATIONS OF EVENT HANDLING

E.E. McDonnell
I.P. Sharp Associates, Inc.
Palo Alto, California

Abstract

The event handling facilities in Sharp APL are relatively new. This paper describes some interesting applications of these facilities, sometimes in ways that are not obvious.

Introduction

Several uses of the event handling facilities are described in this paper:

- input validation
- workspace full handling
- special use of the break key
- handling non-APL terminals
- simplifying file operations
- application system monitoring
- writing secure systems
- extra-syntactical facilities
- fault diagnosis
- dyadic execute model

Summary of Event-handling Facilities

There are two system variables and a system function.

⎕TRAP This system variable can be set to tell the interpreter what events the user wants to handle, and what action is desired if the event occurs.

⎕ER This system variable can be referenced to determine what kind of event occurred, and what was in execution when it occurred.

⎕SIGNAL This system function permits a user-defined function to abort its own execution and generate an error signal, in the same way that the interpreter may abort execution of a primitive function.

Examples

The examples which follow begin with some fairly obvious uses and get more complicated (though still useful) in the latter part of the paper.

1) Input validation

In a large system developed for an airline, the user didn't want to be in immediate execution, but did want freedom to control what was to happen next. There was thus a lot of quad-prime interaction, and many different errors were possible resulting from invalid input. These were not APL errors, but mistakes in the use of the application, which required the application to be written to include many error messages, prompts to the user, and "help" messages to explain to the user what was going on and how to recover.

Before event handling, these requirements frequently added an undue amount of complexity to the structure of the program, often making it difficult to discern its principal purpose. For example, the result of a function would be constrained to serve two purposes: one to give the desired result when all went well, but another to encode in some way the fact that the program hadn't been able to function properly, perhaps because of some user input error. With event handling, the application developer is able to pretend, as far as the result and the structure of the program is concerned, that the user never makes mistakes. When the input parser runs into something it can't deal with, it uses *⎕SIGNAL* with an appropriate message; the top-level function traps the event, issues the error message, and returns control to the point where the user gets to make choices.

This is the type of use which the event-handling facilities were primarily designed to handle. Their benefit here is that they permit the working function to be written more simply, with the burden of negotiating with the user kept in the overall function, which is in a better position to do this.

2) Workspace full handling

In one application where there was a great deal of file record manipulation, as well as movement of functions (in packages) from files on demand, it was possible to write the system so that it remained unaware of the possibility of workspace full occurring. This situation was provided for by use of a global trap which gave control to a cleanup function if a workspace ran out of space.

In this case, as in the input validation case, the gain from having the event-handling facilities was in the greater simplicity of the structure of the main programs. They could be written without worrying about a situation which would rarely occur. Thus they are easier to write, to debug, to document, and to maintain.

3) Special use of the break key

A "menu" type of application is one in which the user is presented with a choice of things to do, and selects one, which the application then does. In one such menu application, one of the choices involves the display of fairly long pieces of text. The user is able to discontinue display of the text and revert to the menu state by hitting the break key. The trap is set so that on the occurrence of a break signal control is passed back to the level of the menu-displaying function, by using the C, or "cut back" option. Display of the output stops, and the user is able to choose a suitable next action.

4) Handling non-APL terminals

It has been difficult for a user of APL at a non-APL terminal (for example, a teletype) to work with an APL application, because if an interrupt occurs, there is no way to key in →⎕*LC* -- the '→' and '⎕' characters don't exist on the keyboard, and can't be transmitted. In a crew-scheduling application, this problem was solved, using event handling, by including a *RESUME* function in the workspace. The definition of this function was:

```
     ∇ RESUME
[1]    ⎕SP←⎕PACK '⎕SP ⎕TRAP ⎕ER'
[2]    ⎕TRAP←'∇ 550 E ⎕PDEF ⎕SP ◇ →⎕LC'
[3]    ⎕SIGNAL 550
     ∇
```

It might be worthwhile to discuss what goes on in the function in some detail. In the first line, the current values of the three variables ⎕*SP*, ⎕*TRAP*, and ⎕*ER* are packaged and made the new value of ⎕*SP*. In the second line a new value is given to ⎕*TRAP*. The number 550 is given as the number of the event to which this trap definition applies; the action code E signifies that when event 550 occurs the statement ⎕*PDEF* ⎕*SP* is to be executed, followed by the statement →⎕*LC*. We'll get back to what this does after noting that line 3 signals the occurrence of event 550, and thus terminates the execution of the *RESUME* function. Now the interpreter inspects the trap statement and executes its two parts. In executing the first part, it causes the variables ⎕*SP*, ⎕*TRAP*, and ⎕*ER* to resume their previous values. In executing the second part, it causes the interrupted function (whatever it was) to resume, just as if →⎕*LC* had been entered from the terminal keyboard. The net effect is that someone at a non-APL terminal may nonetheless be able to resume an interrupted function, even though the '→' and '⎕' keys don't exist on the keyboard.

5) Simplifying file operations

There are circumstances associated with the use of the file system which a prudent application programmer will want to pay heed to, but which, if they accompany every file operation, would make the programming for these operations longer. What I have in mind are such circumstances as *FILE FULL* or *FILE SYSTEM NOT AVAILABLE*. These don't occur very often, but when they do it could be very inconvenient. Since these are both trappable events, it is possible to write functions like the following to do the desired file operations:

```
     ∇ A REPLACE B;⎕TRAP
[1]    ⎕TRAP←'∇ 21 C ⎕TRAP←⍳0 ' ⍝ FILE FULL ERROR
[2]    ⎕TRAP←⎕TRAP,'◇ (5E5+⌈/2↑2↓⎕SIZE 1⍴B)⎕RESIZE 1⍴B '
[3]    ⎕TRAP←⎕TRAP,' ◇ →⎕LC' ⍝ RESUME EXECUTION
[4]    A ⎕REPLACE B
```

In this function we see that the event trapped is *FILE FULL* (event 21). If the ⎕*REPLACE* should be interrupted because the file is full, the recovery expression is executed: first the ⎕*TRAP* variable is set empty (this is to avoid infinite loops); next, the file is resized (by a fairly healthy amount, to minimize the number of times this relatively expensive operation must take place); and last, the line which was interrupted is re-executed. This kind of function could be written for each of the file primitives; the trap could be elaborated to account for each of the interrupts associated with file operations -- but the basic idea will remain the same. For example, if the event were *FILE SYSTEM NOT AVAILABLE*, the application would report this, and give the user the option of waiting or terminating.

6) Application system monitoring

The MABRA system is very heavily used, and by now has been fairly well debugged. Its designer, however, wishes to know if an error occurs, so that it can be fixed as quickly as possible. The design includes for this purpose an interesting use of event handling.

If a user of MABRA should run into an error, such as a syntax error, the standard error message is printed -- that is, no attempt is made by the application to find out why the error occurred. However, the application does send an automatic MAILBOX message (submitted from the MABRA development number through an N-task) giving details of who was using MABRA when the error occurred, when it occurred, what the state indicator looked like, and what files were in use. This message is sent to three people familiar with the internals of the application and usually the problem can be sorted out in a few hours. The usual types of error that have been reported by this means are file reservation and file quota used up. To send a mailbox message the user's T-task must start an N-task on the developer's account, and must therefore have the developer's sign-on information. This must be kept secret. The next section describes a mechanism for doing this, using the event-handling facilities.

7) Writing secure systems

A defect of many techniques that have been proposed in developing secure systems is that a user may break the confidentiality of a workspace by interrupting before the first line of a function in a latent expression can be executed. The function may be performing validation checks or using passnumbers on a file or using a sign-on password as part of ⎕*RUN*, for example. A technique is available, using event trapping, that allows secrecy to be maintained.

It is best explained in terms of an example. The application programmer writes a function similar to the following:

```
      ∇ FRIEND;⎕LX;⎕TRAP;SECRET
[1]     ∘
[2]     ACTON SECRET
      ∇
```

This strange function is now executed by the application programmer.

```
FRIEND

SYNTAX ERROR
FRIEND[1] ∘
          ^
```

As you would expect, the function stops on the first line. Note that there are three local variables associated with the suspended function:

```
)SIV
FRIEND[1] *     ⎕LX     ⎕TRAP   SECRET
```

now the programmer assigns values to these:

```
        SECRET←'ABSOLUTELY SAFE'
        ⎕LX←'→2'
        ⎕TRAP←'∇ 1000 E →'
```

and finally saves the workspace, with the function suspended at line 1:

```
        )SAVE SECURE
```

Since the variables containing the confidential information are local, they can't be obtained by using *)COPY*, so let's see what happens when someone using *)LOAD* tries to break the security by using the break key.

```
        )LOAD SECURE
  \ | /
--BREAK--    (user instantly pounds on break key)
  / | \
```

The latent expression is set to cause execution of the interrupted function *FRIEND* to resume on line 2. Before the latent expression can be executed however, the quick-fingered user has hit the break key. This causes an interrupt. But the trap expression indicates that if **any** interrupt (event number 1000) occurs, the naked branch should be executed -- and this causes the function currently being executed (*FRIEND*) to be aborted. This of course, removes all trace of the variables that were local to *FRIEND*. So now, even though the function *FRIEND* is unlocked, the user is unable to determine what values were associated with its three local variables, and the security of the application is preserved.

8) Extra-syntactical facilities

It has been possible to use the event-handling facilities to model extensions to APL. One such extension is a direct-definition facility. A workspace was prepared which had a trap expression set to handle syntax errors. When a user entered an expression such as:

```
        F:α+÷ω
```

this caused a syntax error to occur. The trap expression called into execution a function which analyzed the second line of *⎕ER*. The function was able to recognize the direct-definition form. If it found that the syntax error had been caused by such a form, it defined the function associated with the form. Of course, if the syntax error was not of this form, the user was merely informed that an error had taken place in the usual way.

This is a fairly powerful facility, and not only allows modelling of new language extensions like this, but also permits applications to invent syntactical rules all their own. In the crew-scheduling application, the convention is used that *?* may be used to elicit help. This works fine inside an executing function, but would ordinarily be embarrassing if the user did this while in immediate execution mode. To avoid such problems, a global trap is set which looks at syntax errors to see if what caused it was a use of *?*. If this was the case, instead of notifying the user of the error, a help message, with a short menu of possible actions, is displayed instead.

9) Fault diagnosis

An unexpected benefit of the event handling facilities is the availability of the information in ⎕*ER* after every interrupt. We had a telephone call in our office from someone who couldn't understand what had happened in an N-task that he had started, but which hadn't finished properly. He described the application and nothing in what he said gave a clue to what might have gone wrong. Finally it occurred to me to ask what the value of ⎕*ER* was. With this piece of information everything became clear, and it was easy to straighten the user out.

10) Dyadic execute model

One of the experimental language extensions that has been modelled using the event handling facilities is one which itself deals with event handling. This is a proposal for a dyadic execute primitive with the following syntax and semantics:

```
        A⍎B
```

A and *B* are character vectors. *B* is executed if possible, otherwise *A* is executed. *A* is also executed if *B* is empty. In particular, if the execution of *B* would cause an error report, *A* is executed instead. If the execution of *A* produces an error, the error is reported. A function which models dyadic execute is given below.

```
      ∇ Z←L DEX R; ⎕TRAP; ⎕ER
 [1]   →(' '∧.=,R)/6
 [2]    ⎕TRAP←'∇ 0 C ⎕TRAP←⍳0 ◇ →5'
 [3]    Z←⍎R
 [4]   →0
 [5]   →((6=1↑⎕FI,⎕ER)∧'DEX[3] '∧.=⎕ER[1+⎕IO;⍳7])/0
 [6]    ⎕TRAP←'∇ 0 C ⎕TRAP←⍳0 ◇ →9'
 [7]    Z←⍎L
 [8]   →0
 [9]   →((6=1↑⎕FI,⎕ER)∧'DEX[7] '∧.=⎕ER[1+⎕IO;⍳7])/0
[10]    ⎕ER
[11]    ⎕SIGNAL 11 ⍝ DOMAIN ERROR
      ∇
```

For many error trapping problems, the function *DEX* provides a simple and powerful solution. The main complication in the function, at label *RE* and again at *LE*, is to distinguish errors in executing the expression from the value error caused by the expression not having a result. Resetting ⎕*TRAP* to be empty in statement *LL* is a very useful technique, as it can prevent trapping errors in the code handling the original event.

Events That Can't Be Trapped

The following information is taken from the Sharp APL Reference Manual.

Certain errors and interrupts can't be trapped. They are the following:

1. An error encountered while executing the line that is part of the trap definition. An error in the trap line is exempt from trapping, primarily to avoid endless recursions that might otherwise arise from a poorly chosen trap line.

However, an error in a function invoked by the trap line, or in the argument of ⍎, **can** be trapped in the usual way.

A function invoked by the trap line but aborted by ⎕*SIGNAL* can also be trapped. In that case, ⎕*ER* reproduces the trap statement.

2. An interrupt caused by ⎕*BOUNCE*. When a task is "bounced", provided *CLEAROUT* has not been set, its active workspace is saved. ⎕*ER* in the saved workspace will show one of the interrupts. Which one depends on the circumstances when the task was bounced.

3. Any of the following errors is considered an error in transmitting or setting up the line for execution, rather than an error in execution. None of them causes ⎕*ER* to be set. None of them has an event number.

OPEN QUOTE The line contains unbalanced quotes.

CHAR ERROR The line as received from the terminal contains an unrecognizable character. The system displays the characters preceding the first offending character, and awaits a corrected reply.

WS FULL ERROR There is insufficient space to prepare the line for execution. **Note**: This message is distinct from event 1 *WS FULL*, which indicates that there isn't space to complete execution of the line.

SYMBOL TABLE FULL ERROR There is insufficient space in the symbol table to prepare the line for execution. **Note**: This message is distinct from event 15 *SYMBOL TABLE FULL*, which indicates that the symbol table lacks space to accommodate names generated during execution of the line.

DEFN ERROR The line contains an improper use of the ∇ character.

Acknowledgements

I received an enormous amount of help in the preparation of this paper from the following people at I.P. Sharp Associates: Paul Berry, Doug Forkes, Simon Garland, Lib Gibson, Hugh Hyndman, Eric Iverson, Keith Iverson, Paul Jackson, Bob Metzger, Roland Pesch, Roberto Skertchly, Ed Stubbs, Joey Tuttle, Rosanne Wild, and Peter Wooster.

AN APL CORPORATE MODEL FOR AN ELECTRIC UTILITY CORPORATION

Charles O. Manahan
Southern Company Services
Atlanta, Georgia

Southern Company

The Southern Electric System consists of the parent company, The Southern Company, four electric power operating companies, and a completely owned service organization, Southern Company Services. The four power companies — Alabama Power, Georgia Power, Gulf Power and Mississippi Power — are all located in the Southeastern United States. The Southern Electric System is one of the most widely owned stocks in the United States. Today it is owned by more than 341,000 common stockholders throughout the world. Together, these companies employ more than 26,000 people and serve more than 2.5 million customers in a service of about 120,000 square miles. Our service area includes most of Alabama and Georgia, the northwest portion of Florida and the southeast section of Mississippi. Our combined system has over 22,500 megawatts of generating capacity. Southern Company Services, with offices in Atlanta, Birmingham and New York, provides technical and other specialized services to the parent company and the four power companies. Included in these services is operation of a centralized Data Center. Centralization of Data Processing equipment and development of standard information systems has enabled our companies to realize data processing economies and efficiencies. More than 829,700 computer jobs were processed by our Data Center in 1979.

Short Model History

A corporate planning process has existed within the Southern Company for some time. It is the process by which a corporate budget/or five year forecast is prepared once each year. It is a lengthy, comprehensive process that is highly decentralized, involving many functional areas in the power companies and Southern Company Services.

The major forecasting areas that make up the total Corporate plan are:

- An Economic Forecast
- A Load and Energy Forecast
- A Fuel Forecast
- A Facilities (or Generation) Plan
- A Revenue Forecast
- The Operating and Construction Costs Forecast
- Financial Projections

Many of these forecasts have been computer aided, but some have always been prepared manually and all require a considerable amount of engineering and management judgement. The corporate planning process was very accurate and reliable until the early '70's. Southern Company Load Forecasts were based upon manual trend-line methods, where future energy sales and peak demands were extrapolated from historical data. Expansion of generation facilities could be planned with a high degree of confidence. However, changes occuring in 1973 and '74, beginning with the first energy crises, demonstrated that the old procedures could not be entirely relied upon.

To cope with the new planning environment, two corporate planning capabilities were developed. First a strategic planning process was introduced to develop alternative strategies for dealing with assumed conditions in the Southern Company's future. The strategic planning process emphasized longer range planning — beyond the 5 year budget forecast. It emphasizes examination of many options and communication with the key people throughout the organization. To assist in this strategic planning process, management asked for development of a capability which could analyze a broad range of questions yet be simple enough to avoid many man-years of development effort. Thus, the second corporate planning capability was developed — the Corporate Model.

APL (A Programming Language) was selected as the computer language for the model. There were several reasons for this choice. (1) APL is an interactive language, and the model is, and was intended from its inception to be, an interactive model. (2) APL program development time is usually more rapid than development in other languages. (3) Finally, APL has an array manipulation capability which is superior to most other standard computer languages.

How Model is Used at Southern

The Corporate Model's use is solely for strategic, long range planning and rapid comparison of alternate strategies. The model can analyze questions in the entire range of the planning process, but its emphasis is always on the evaluation of the relative merits of various planning assumptions. It is not a tool for the detailed analysis upon which a final decision should be based.

Since the model is supposed to be used as a rapid analysis tool, its use is decentralized. There is no single group which controls the model. A user team of 5-8 people is designated at each operating utility, and these groups use the model according to the needs of their management. This decentralization necessitates a model which can "stand alone" and can be used without requiring a lot of specialized information such as a programming language, storage locations, etc. To allow as wide as possible a usage, the model is a completely prompted, totally interactive structure. Absolutely no programming is necessary to examine a wide variety of strategic options with the model.

The model must have as many internal checks and controls as feasible since there is no single organization responsible for ensuring the accuracy of input, lack of contradiction in assumptions tested, etc. More code is devoted to managing the model's input and outputs and to internal validity checking than is actually used to calculate results.

Model Subsystem Overview

The part of the model which calculates the results is subdivided into six sections which simulate key utility planning areas. These are:

- Load and Energy Forecasting
- Facilities for Generation Planning
- Production or Operating Costs
- The Revenue Forecast
- Construction Costs
- Financial Projections

Each one of these is a Corporate Model subsystem with the following objectives:

> The Load and Energy Forecasting Subsystem provides the annual energy demand and sales forecast for each operating company and the total Southern Company system. It also projects the number of customers by customer class for each company. All forecasts of the model are expressed in terms of annual data only.
>
> The output of the Load and Energy Forecasting subsystem defines the amount of service our system must provide and is therefore very important to the facilities planning or Generation Planning phase. The Generation Planning subsystem is essentially a file maintenance activity that guides the user in describing the amount and type of generation needed to satisfy the load requirements of the load forecast.
>
> The cost of producing the energy required by the load forecast, (using the type of production facilities described in the generation plan) is the function of the Production Costing subsystem. It calculates the fuel, plant operating and maintenance costs, the amount and cost of energy interchanged within our system and the amount and value of generating capacity bought from and sold to the Southern Company's capacity pool.
>
> The costs of constructing the generating facilities specified in the generation plan are provided by the Construction Costing subsystem. It calculates annual construction costs for all new generation, production modifications, transmission and distribution and other items, such as nuclear fuel start-up costs and construction of step-up substations.
>
> The Revenue Forecasting subsystem provides annual base rate revenues for each customer class based on an average rate per KWH for each customer class **or** based on rate schedules for each customer class. The primary information for the revenue calculation is amount of KWH sales, KWH demand and number of customers as forecasted by the Load and Energy subsystem.
>
> Based on outputs from all the other Model's subsystems, the Financial subsystem provides a financial analysis in terms of a Balance Sheet, Net Income Report, Sources and Uses of Funds and a Special Management Analysis Report.

There are two additional subsystems of the Model which are not used directly in computing results. These are the Session Supervisor and the Data Management subsystems.

The "Supervisor" subsystem provides the conversational interface between the user and all other parts of the Model. It is the operating system under which the Model runs. It is present at all times during a modelling session and performs the following tasks:

- Session organization and coordination
- All general utility functions
- Security checks against invalid use of the Model
- All data file manipulations

The Supervisor controls the interaction of all other subsystems of the Model including a Data Management Subsystem. The Data Management Subsystem is a fully prompted routine to add, change or delete data in preparation for use in one of the six arithmetic processing subsystems. Its functions are limited to the following capabilities:

- Manipulation of data values and data descriptions
- Creation of new data files
- Modification of existing data files
- Establishing data value validation rules

It performs no arithmetic operations.

The above gives the scope of our Corporate Model. The following is a more detailed exposition of how the Model operates.

Load and Energy Forecasting

This subsystem does not **develop** a load forecast. It provides a **changed** load forecast using pre-established energy and peak loads for each company and the Southern Company system, pre-established numbers of customers by class and historical load duration curves as a starting point. The pre-established figures consist of the latest approved forecasts for each company. The user specifies changes to the pre-established forecast to produce a new forecast.

For example, the load duration curve in this subsystem is divided into 24 energy bands. Each energy band has 19 characteristics, such as summer day time energy, summer night time energy, winter day time energy, etc. User specified changes to these 19 characteristics cause the energy in each band to be re-calculated and a new load curve is produced. Coincident load curves for each company are used to derive the Southern Company system load curve.

The amount of energy sales and number of customers forecasted are transferred, by the Supervisor subsystem, to the Revenue and Financial subsystems. The load curves for each company, for each year of a study, are transferred to the Production Costing subsystem along with a matching Generation plan.

Generation Plan

As in the Load and Energy Forecasting subsystem, the Generation Planning subsystem does not develop a facilities plan. A new plan is provided by **changing** a previous plan. A user adjusts the previously defined generation expansion plan for generator and production modification additions or deletions, construction delays or advances and for changes in generator operating characteristics. There are almost an infinite number of changes that can be made to a generation plan, but none require arithmetic operations. Therefore, this subsystem is a data file maintenance tool only.

The new generation plan provided by this subsystem is used by the Production Costing and Construction Costing subsystems.

Production Costing

This subsystem uses the load and energy forecast and generation plan data to perform five major calculations:

1. It calculates the territorial generating capacity for each company in the Southern Company system.

2. Provides annual production, operating and maintenance costs for each generator. These costs are accumulated by company, type of generator and by generator location.

3. Southern Company system energy interchange requirements for each generator, including fuel and total variable costs for the energy interchange.

4. Cost of energy sold to and purchased from the Southern Company system energy pool.

5. Amount and cost of emergency purchases from outside the Southern Company system.

This subsystem uses the "stacked de-rated dispatch" method of scheduling the use of each generator in a generation plan. This is a method of determining the amount of each generator's energy production used to meet the energy requirements in each of the energy bands of the load duration curve based on each generator's cost of operation. Each generator's known capacity to produce energy is reduced (de-rated) below its normal rating to account for contingencies, such as unscheduled maintenance. This stacking of generators within each energy band of the load curve allows the selection of the least costly generator to meet the energy requirements at each point. Thus, least expensive energy is always produced first and the most expensive is produced (or purchased) last in the Model.

The fuel costs and amount of energy sales to certain wholesale customers are transferred to the Revenue subsystem. All operating costs plus the amount and cost of energy and capacity bought and sold within the Southern Company system, are used by the Financial subsystem.

Construction Costing

The generation plan described earlier is the primary determinant of the amount of funds expended for construction. Expenditures for constructing new generators and their associated facilities depend upon the number and type of generators described in the generation plan and upon standard construction costs for groups of generators that are similar. Adjustments in construction costs for delays or compressions in the construction schedule and joint ownership of generators are automatically handled.

Expenditures for construction projects already underway are calculated by adjusting the forecasted expenditures for any costs already incurred. This is the main purpose of the "historical cost data" input file.

Construction expenditures for other cost categories, such as, transmission and distribution facilities, production modifications, nuclear fuel contracts and other general construction are calculated from separate forecasts supplied by users through the "associated facilities" data.

Any of the costs calculated by this subsystem can be adjusted to account for special situations beyond the normal capabilities of the subsystem. These adjustments are input in the "$adjustments" data file and directly change the final calculation of construction costs.

The construction costs provided by this subsystem are used by the Financial subsystem.

Revenue Forecasting

The Revenue Forecasting subsystem provides the option of calculating annual revenues by two different methods. One method uses a simple "average rate" approach where an average rate per KWH is specified for each class of customer. In this method the amount of KWH forecasted by the Load and Energy subsystem is multiplied by the average rate specified for each customer class to give the forecasted revenue.

The other approach calculates a KWH usage per customer vs. numbers of customers curve, divides this curve into intervals representing various rate schedules and determines the area in the various intervals. This area is multiplied by the appropriate rate schedule to determine the revenues for each separate schedule. This approach allows more flexibility in asking "what if" questions by simply respecifying breakpoints in the rate schedule.

The revenue forecasted by this subsystem is transferred to the Financial subsystem. This is the last input required from the Model's subsystem before the financial analysis can be produced.

Financial Planning

Based upon inputs from all the Model's other subsystems the Financial Planning Subsystem provides forecasts of:

- Fuel clause adjustment revenue
- Operating and Maintenance expenses
- Depreciation
- Income Taxes
- Dividend Payments

These calculations are made using financial constraints specified by users of this subsystem.

The financial projections are presented in formats familiar to management, such as:

- Income statement
- Sources and Uses of Funds
- Balance Sheet
- Management Analysis Report
- Plan-to-Plan Comparison of selected financial items

All these projections are made on an aggregate annual basis rather than on a detailed account-by-account basis. Therefore, the results are not expected to be accurate enough upon which to base a final plan, but may be used to rapidly select a few alternatives for detailed study.

Scenario Concept

The single most important user control in the use of the Southern Company Corporate Model is the scenario concept. It is possible to have any number of "correct" input data files to a given subsystem of the Model, all differing significantly from each other. The problem with this is how to identify the data files associated with various planning assumptions being tested by the Model and how to associate groups of related data files for the different Model's subsystems.

The solution to this problem is the Scenario Catalog. The Scenario Catalog is an on-line character image file which contains a five digit scenario number, a scenario title, date of creation, descriptive material, and a listing of what specific data files were used as input to and output from each scenario. All data files which are permanently stored have, as the last five characters of the file title, the five digit number of the scenario for which they were created. Whenever a data file is used without modification, it retains the scenario number for which it was created. In order to identify what planning assumptions are the basis for any given data file the user only has to call the scenario description which matches the last five digits of this data file title.

When a planner uses new or modified data for the first time, he is allowed two dispositions of the output. If, after examination of results he is satisfied, he may save the data file as part of a scenario. In this case, all output data files created and any new or modified input data are given new titles with the new scenario number as the last five characters of the title. He may, of course, choose not to save anything in which case nothing is retitled in the Scenario Catalog.

Thus, the Scenario Catalog provides the Southern Company Corporate Model user an on-line cross reference to his planning assumptions and associated data files.

This catalog is also read by the Supervisor Subsystem as part of the internal checking that is done every time the Model is used. For example, if a user attempts to save output from the Production Costing subsystem into a scenario which already has Production Costing data files, the Supervisor Subsystem notes this by reading the Scenario Catalog, the user is given an error message, is forced to save his output elsewhere, and is required to enter additional descriptive material.

File Management via Supervisor

One of the primary design parameters of our Corporate Model is that it was to be conversational as well as interactive. The user does not need to know any programming language to use the Model. The user may invoke the entire range of Corporate Model responses by simply responding to prompts. This is accomplished by imbedding all of the Model's programs in a single control program called the Supervisor Subsystem.

All Corporate Model data files are APL workspaces. This allows data to be stored in the form used by the processing programs and eliminates the need for any special data conversion programs. The Model uses data structures from scalars through 5 dimensional array. The APL workspaces are manipulated by means of the standard APL system commands (i.e., *SAVE*, *COPY*, etc.). These commands are incorporated under program control by means of auxiliary processors. The specific auxiliary processor used is the stack input processor, IBM auxiliary processor 101. The Session Supervisor builds a stack of commands and then halts normal processing until the stack is executed. A typical stack to save an output file will look as follows:

```
a)    )SAVE TEMPWS
b)    )CLEAR
c)    )COPY TEMPWS OUTGROUP
d)    )SAVE NEWNAME
e)    )LOAD TEMPWS
f)    )DROP TEMPWS
g)    →suspended line
```

a) This saves a temporary copy of the entire workspace.
b) This clears the workspace.
c) This copies a specified predefined group of output variables into the cleared workspace.
d) The output variables are saved under a new file name.
e&f) The temporary workspace is reloaded and then dropped.
g) Normal processing is resumed by branching to the suspended line.

The Model uses three IBM 3330 disk drives as primary data storage and part of an IBM 3851 Mass Storage Subsystem to store older files. As a result, there are a large number of possible storage locations for files; furthermore, all of this storage is password protected to prevent inadvertent erasure or unauthorized use. The Supervisor Subsystem determines the storage location by an algorithm based on the user's company affiliation, type of processing used, and file age. This algorithm is used to access the proper storage location and copy the file into the active workspace (or save the workspace to the storage location in the case of file creation). All files over a certain age are stored on an IBM 3851 Mass Storage unit.

The Southern Company's Corporate Model runs under IBM's VM (Virtual Machine facility). The mass storage unit is accessible only under MVS (Multiple Virtual Systems), a completely separate set of CPUs from the VM system. File movement from VM to mass storage and vice versa requires the submission of a JCL (Job Control Language) card deck (or disk file equivalent) for every file moved. This is a tedious process to do manually. When a Model user requests a file that is on mass storage, he simply supplies the file title (8 characters) in response to a prompt. The JCL input is written and submitted to the batch processing CPU automatically by the Supervisor Subsystem. The Supervisor periodically examines the batch systems' output for the requested file, and when it arrives the file is moved to a location accessible to APL and the Corporate Model. The user is then informed via the Supervisor Subsystem that his file is available and Model processing continues. The average time to retrieve a file from mass storage is 25 minutes.

To avoid annoying delays, files which are returned from mass storage are placed in a special disk storage location of limited size. Files are not removed from this location unless the space is needed by new incoming files. In this case the least frequently used files are erased to make room for the new ones. This completely eliminates the retrieval from mass storage for old but frequently used files.

Corporate Model Maintenance

The Southern Company's Corporate Model is constantly being changed to reflect changes in company operation, changes in the types of studies done with the Model, to enhance Model accuracy, and to repair errors. APL readily lends itself to easy maintenance. We frequently repair or change the model from verbal descriptions. Our experience with the Model has shown that most changes and error corrections can be completed within a few hours of the request. The APL error messages and the ability to repair a function line while the function is suspended are the prime contributions to the rapidity with which APL code can be changed and debugged. APL enables us to have a much more responsive model than would be the case if it were programmed in another language.

Model Security and Controls

The primary model control is, as previously mentioned, the Scenario Catalog. However, with the large number of users in the four operating companies of the Southern System, additional controls are necessary. It is necessary to restrict access to company specific files to those individuals who have been authorized by the particular operating company. Individuals are restricted both in which processing subsystems of the Model they can access and what level of use they may have. We have several usage or access levels for every processing subsystem of the Model. The highest level is complete access with both logic and data change capability. The next level is data change only, and so forth down to clerical levels which may only print reports.

The finely subdivided information as to who may do what in the Model lends itself very readily to being stored in an APL multidimensional array. This is indeed how it is stored. All user numbers, operating companies, processing subsystems, and access levels are stored in a single four dimensional array. The Session Supervisor merely indexes to the proper position in this array to determine whether a person has permission to do an operation. APL's indexing and array handling capabilities have made trivial what could have been a tedious task, maintenance of Model security.

Variety of Uses of APL Within the Model

One of the common criticisms of APL is that it is an excellent language for mathematical processing, but leaves much to be desired when used for other types of tasks. We find that APL is useful for a wide variety of disparate chores.

In the Model we use APL for

1. Matrix manipulation (Load Forecasting Subsystem).
2. Engineering simulation (Production Costing Subsystem).
3. Standard accounting (Construction Costing and Financial Planning Subsystem).
4. Data Management (Generation Planning and Data Management Subsystem).
5. Operating system interaction and file handling (Session Supervisor).
6. Algorithmic usage (Revenue Forecasting Subsystem).

7. Conversational interaction (Session Supervisor, Data Management and Generation, and Generation Planning Subsystems).

All of these uses are fairly easy to program in APL. In fact, even in the areas of file handling and operating system interaction it is quicker to meet requirements with an APL program than to write a COBOL or FORTRAN routine to do the same thing. For those tasks that APL is not equipped to perform, i.e., storage area movement, punch and printer control, etc., the ability of APL to manipulate character arrays makes it easy to use APL to build a character image file which is an executable program in the operating system language. This program is then executed to accomplish the desired result. These programs can be written and executed under APL program control thus greatly enhancing the flexibility of APL and extending the range of tasks that can be performed by APL.

Problems with APL

In our entire range of model operations we can identify only two areas where APL is not superior to other languages: APL has difficulty in handling very large data structures. For example, our customer accounting files have over two and a half million records with each record being several thousand characters in length. APL can not efficiently manipulate these files. The maximum workspace size available at our installation is about 2.1 megabytes, thus the upper size limit for efficient manipulation of data structures is considerably below this.

The other deficiency is in support. We use IBM APL, and from time to time we have problems which should be handled by IBM. Their response to date has been unsatisfactory. IBM's response is not rapid enough for the modelling environment nor is IBM's usual response very useful when it finally arrives.

PRIVATE NETWORKING IN A MULTINATIONAL

Irene P. Harford
General Manager Office Automation Planning
Massey Ferguson Limited
Toronto, Ontario

Background

Massey-Ferguson is a Canadian based multi-national corporation that manufactures farm machinery and diesel engines.

It has 37 wholly owned factories in nine countries, and along with its associates and licensees, Massey-Ferguson products are made in 85 factories in 31 countries.

Massey-Ferguson products are sold in 190 countries.

With this type of international company structure, and a Head Office based in Toronto, it is essential that information flows quickly and easily among international Operating Units and the Head Office function for all financial information and reporting statistics.

In the sixties, the fastest common communications medium internationally was telex/teletype, and large telex centres evolved, rather than being designed, at main communication centres in North America - such as Des Moines, and in Coventry, England, which was the main communications centre for Europe.

Manual telex refile, using torn tape techniques was the method used to disseminate messages to other Massey-Ferguson and Perkins locations in North America, Europe, and the Southern Hemisphere. Point to point leased lines were installed as traffic costs justified the tariffs involved. Early message switching facilities were rented from the British Post Office (B.P.O.) in the U.K. at London, England, and from Western Union in New York when we cost justified our first momentous quarter speed telegraph circuit between the U.K. and North America.

Network Beginnings

Telex volumes increased rapidly, and the international trunk link of our growing network was increased to half speed in 1973.

However, as our traffic volumes grew, the technical differences of our switching facilities became more apparent. Significant compatibility problems existed between the Philips System offered by the B.P.O. and the Western Union switch. Both systems were progressively upgraded independently but many of the features offered were excluded from our use because of incompatibility in working the systems together.

It is significant to mention at this juncture that at this time the telecommunications function was fragmented within our company structure and did not have a single staff reporting route to a Head Office function. This complicated finding an effective method of reporting problems and failures of our first stage international network, quite simply because it was not identified as a single area of responsibility within our company structure.

A General Manager of Telecommunications was appointed in 1974 reporting to the Corporate Management Systems Director at Head Office and his first task was to produce a solution to the problems of our apparently ever expanding telex traffic demands and inadequate methods of routing and delivering telex messages throughout the network. A consolidated measurement of traffic volumes produced some startling figures.

The cost justification of the use of private message switching facilities in the U.K. became a relatively simple management decision, and similar rented facilities with emulated software were arranged with CNCP in Toronto, Canada. Other Operating Units internationally were added to this network from both the U.K. and Toronto message switches.

A new switching centre was set up in the Coventry area of the U.K. at Stoneleigh in Warwickshire using the ITT ADX 600 for all international port terminations in Europe.

Principally for company domestic and structural reasons, a further ADX 600 was installed at Perkins, Peterborough in the U.K.

Introduction to Time-Sharing

In 1975, Massey-Ferguson introduced I.P. Sharp's APL time-sharing facilities for financial information reporting systems in Europe and North America, initially on a dial-up basis.

APL became the principal financial information reporting system and a means of producing answers to those hypothetical "what if" questions which have to be answered in any corporation.

Time-sharing occupancy times soared rapidly.

In May 1978, processing was transferred from I.P. Sharp's mainframe in Toronto to Massey's Data Centre in Toronto. Concentrator units of the same type as used by Sharp on their own network, the Alpha 16, were installed in Stoneleigh and Peterborough, England; Hanover, Germany; Lucerne, Switzerland; Le Plessis Robinson, Paris, France; Aprilia, Italy; and in Toronto.

Back-up facilities from Sharp were maintained with a link from the Toronto Alpha to Sharp's Alpha in Toronto.

Telegraph, as an ASCII based system with nominal transmission speeds of 50-110 baud was operationally linked with the 300 baud asynchronous APL via the ubiquitous circuit we termed "Circuit X". This circuit theoretically made every telex terminal a time-sharing terminal (if you could put up with the enormous response delay) but more importantly made every APL terminal capable of sending a telex interfacing to the private message switching service.

The trunk carrier circuit between East and West was multiplexed to provide 4800 b.p.s. carrying time-sharing service, and two further channels at 2400 carrying IMS and MVS, using Racal Milgo Multimode 9600 modems at each termination to provide the separation in operating bandwidth.

Rationalisation of Data Centres

Up until this time Massey-Ferguson had been operating its Data Centre Operations on a relatively autonomous basis, with IBM 158's in Des Moines, Toronto, Coventry, Peterborough, Hanover, Paris; a 148 in Rome and an Amdahl V5 in Aprilia, Italy; a 138 in Canton, Ohio, and a 145-500 in Sao Paulo.

A rationalisation of Data Centres, which has significantly increased the operating flexibility and disciplines of a multinational data network, has resulted in two main Data Centre Utilities; one serving North American operations, located in Des Moines, Iowa, and the other serving European locations (with the exception of Perkins, Peterborough) based at Birmingham in the U.K. using an Amdahl V7 in Des Moines and an Amdahl V8 in Birmingham.

The ITT Comten box is used to provide the interface to the mainframe running IMS/MVS at the various locations.

Conversion from local 158's to main Data Centre Utilities is due to be completed by the end of the year which brings us up to date in the evolution of our international data network, which has progressed in 10 years from a very low speed data network to a major private international network with operating speeds of 9600 over two international bearer circuits.

National Voice Network Additions

During the past few years, however, we have also developed and installed two tandem switched voice networks — one in the U.K. and the other in North America.

In the U.K., using Coventry as the tandem switching node, we are using four widebands, i.e., 4 x 48 KHZ, each of which is multiplexed to provide 12 channels for both voice and data; and a channel which can be engineered to further multiplex 10X telegraph channels.

The network is the first private stored programme controlled voice network in the U.K. The terminal PBX's and tandem switch are Plessey PDX which is made under licence in the U.K. from the original ROLM CBX. This is capable of handling local data transmission switching in addition to voice.

The tandem is capable of handling 96 ports, and is at the moment being used for national voice circuits only. Obviously, we wish to include international circuits for voice transmission operating on an auto/auto basis, which is currently constrained because of the problems associated with international agreement on signalling between Europe and North America on private circuits.

We have cost justification to install at least one circuit to Toronto and others to Germany, France and Italy.

The North American voice network is now installed and running using a switch in Detroit; a star network design with some 13 location terminations.

It is planned to link these two networks together. However, international leased lines for voice only transmission, especially when examined as an analogue transmission and the bandwidth required, are extremely expensive on long haul routes, and also give an extremely poor utilisation of bandwidth availability, particularly when measuring the small window of time available during normal working hours between East and West.

It is with this in mind we have been carrying out trials on a speech digitiser box which would allow us to transmit slightly impaired speech transmission over a 2400 bit channel of an international circuit.

Future Network Rationalisation

Our objective, must be to merge all networking applications regardless of type.

This becomes a more obvious essential criteria as distributed processing techniques become more readily acceptable and understood.

Our company lays great emphasis on the development of information processing techniques throughout the organisation, which will provide us in the following years with a single electronic mail system, interfacing to the public networks, as and when they become available; electronic storage; access to and from both private and public data bases; picture as well as text transmission and all systems integration which is generally categorised as office automation.

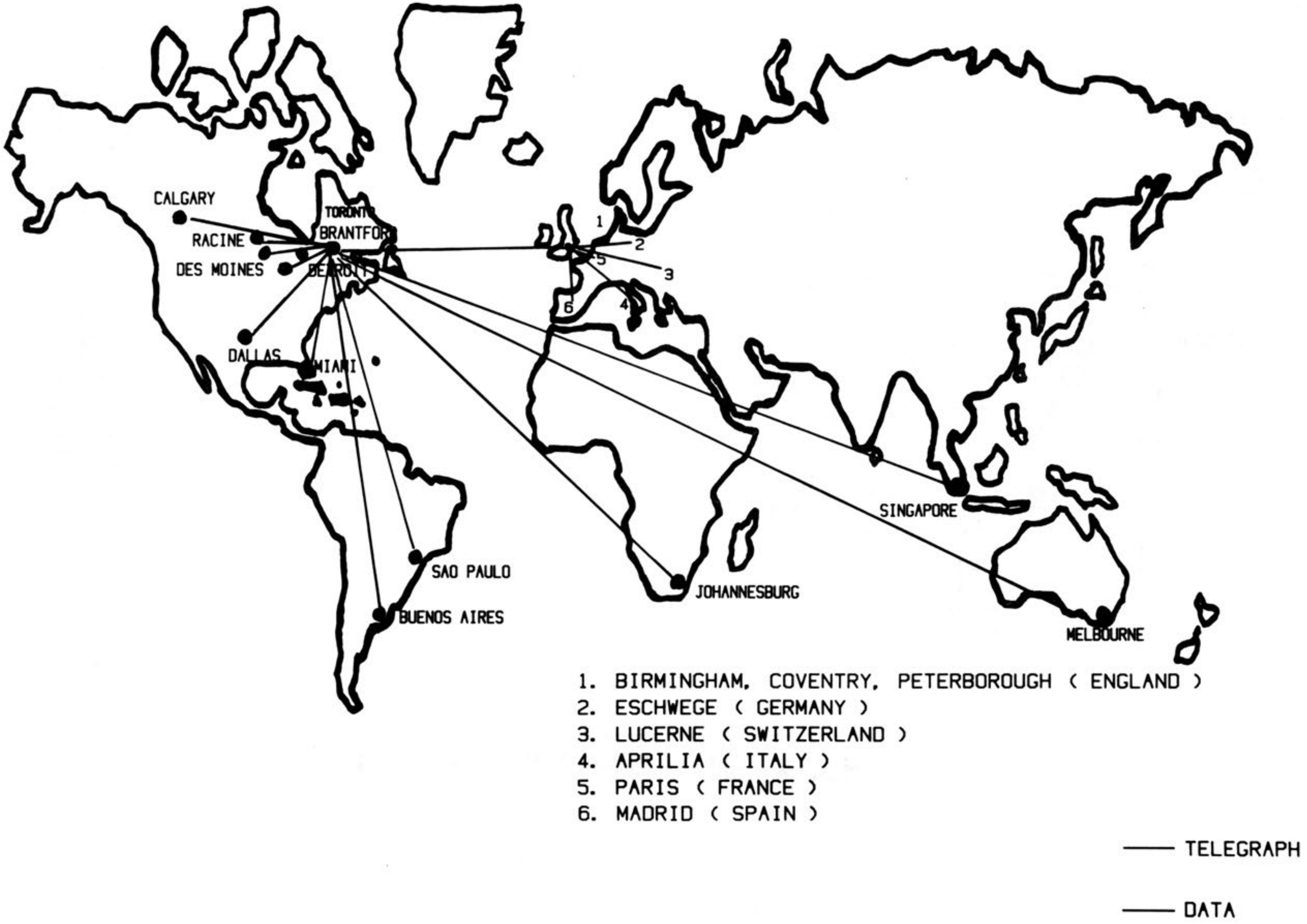
The future phases in the rationalisation of our private international network must consolidate the various stepping stones of switching/concentrator boxes such as the ADX's, the Eclipse, Alpha 16, and Comten, into ideally one such box which is intelligent enough and flexible enough to accept and evaluate various protocols and control functions; has adequate port and bandwidth capability and will accept varying transmission speeds and store and forward.

Similarly we are evaluating the use of terminals. The proliferation of terminals over the past five years is horrendous as terminals have been developed essentially on the basis of one box: one application. In consolidating our systems, this concept becomes blatantly outdated and ludicrous.

We consider it inevitable that there will be many steps to take in reaching our objectives, but we are currently looking at various word processing terminals which have the potential of being used as a common type of terminal and at least initially cover several applications in addition to its basic use as a very smart typewriter.

The potential opportunities operationally and financially are challenging, exciting and without question offer enormous rewards. The next 10 years hold a great deal of promise for a new dimension in networking operations.

MASSEY FERGUSON COMMUNICATIONS NETWORK

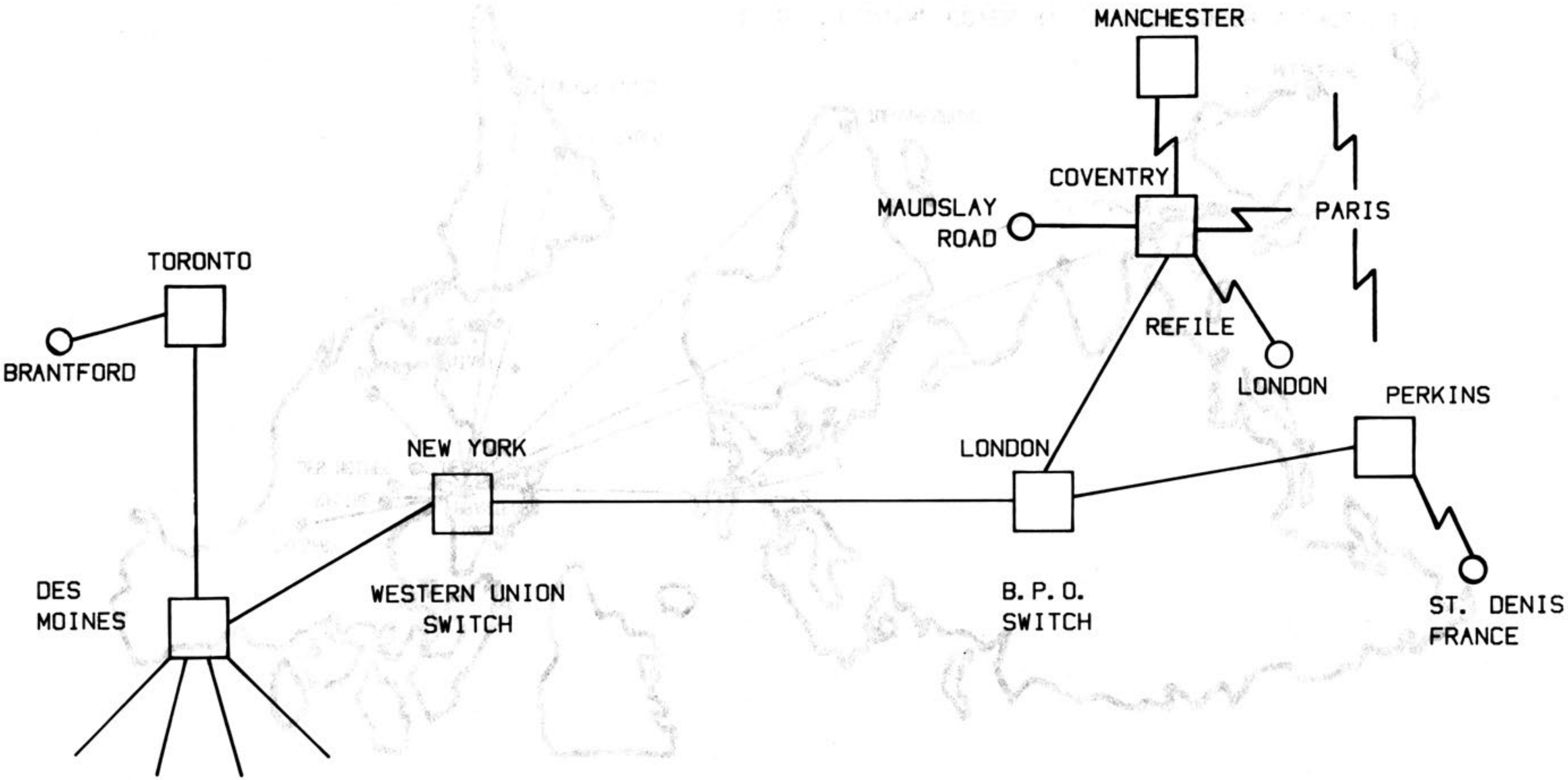


1. BIRMINGHAM, COVENTRY, PETERBOROUGH (ENGLAND)
2. ESCHWEGE (GERMANY)
3. LUCERNE (SWITZERLAND)
4. APRILIA (ITALY)
5. PARIS (FRANCE)
6. MADRID (SPAIN)

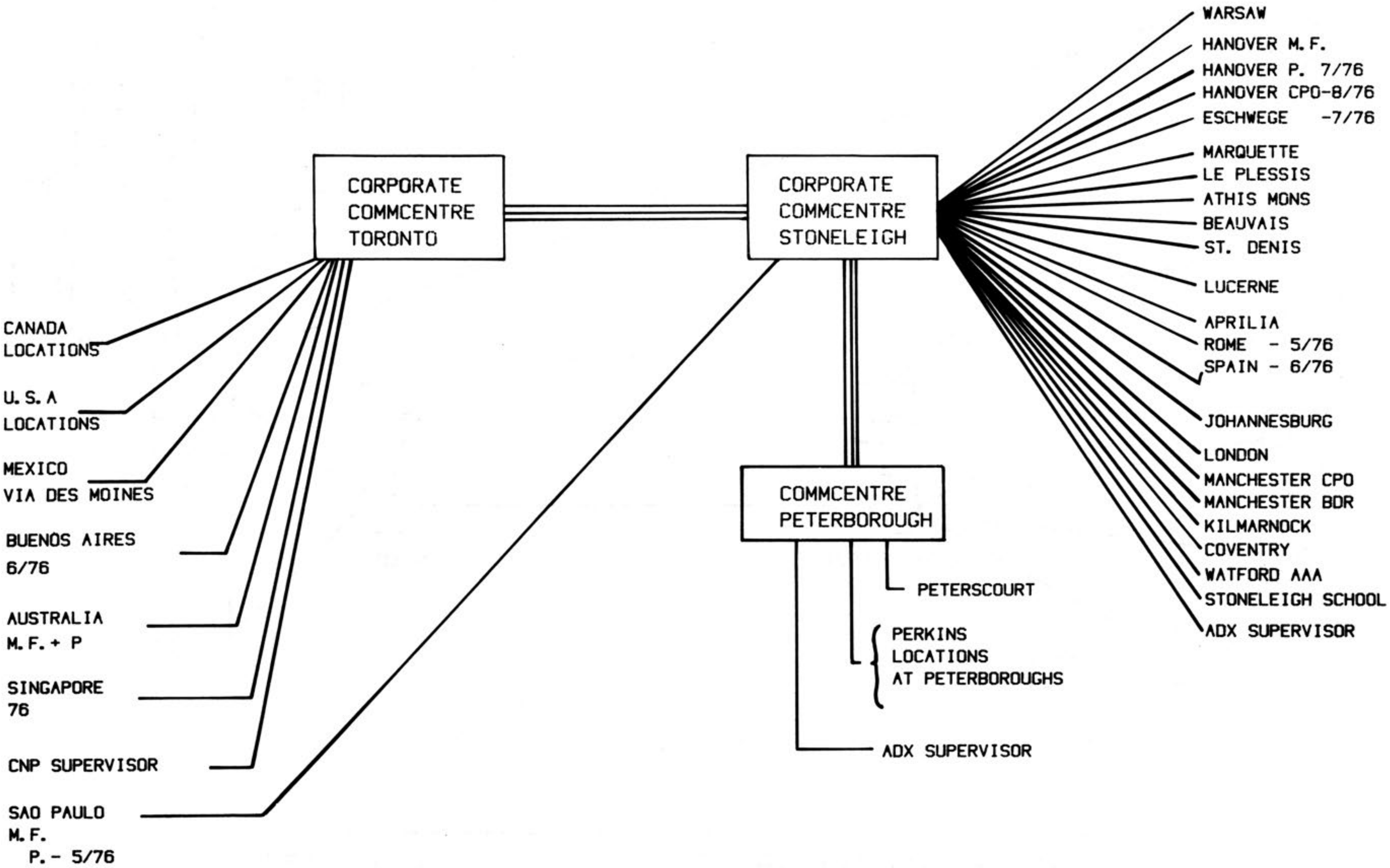
—— TELEGRAPH

—— DATA

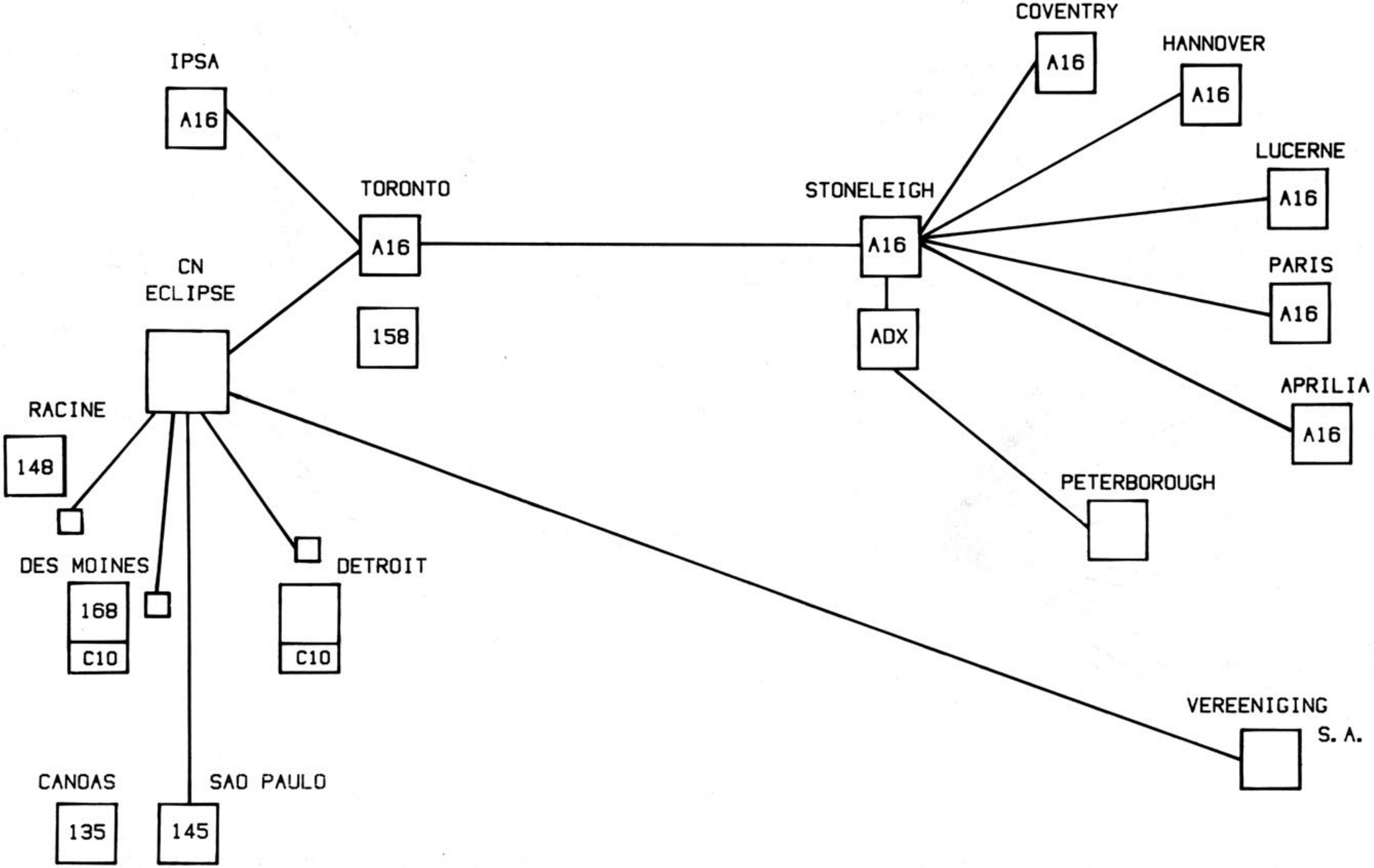
EARLY TELEGRAPH NETWORK



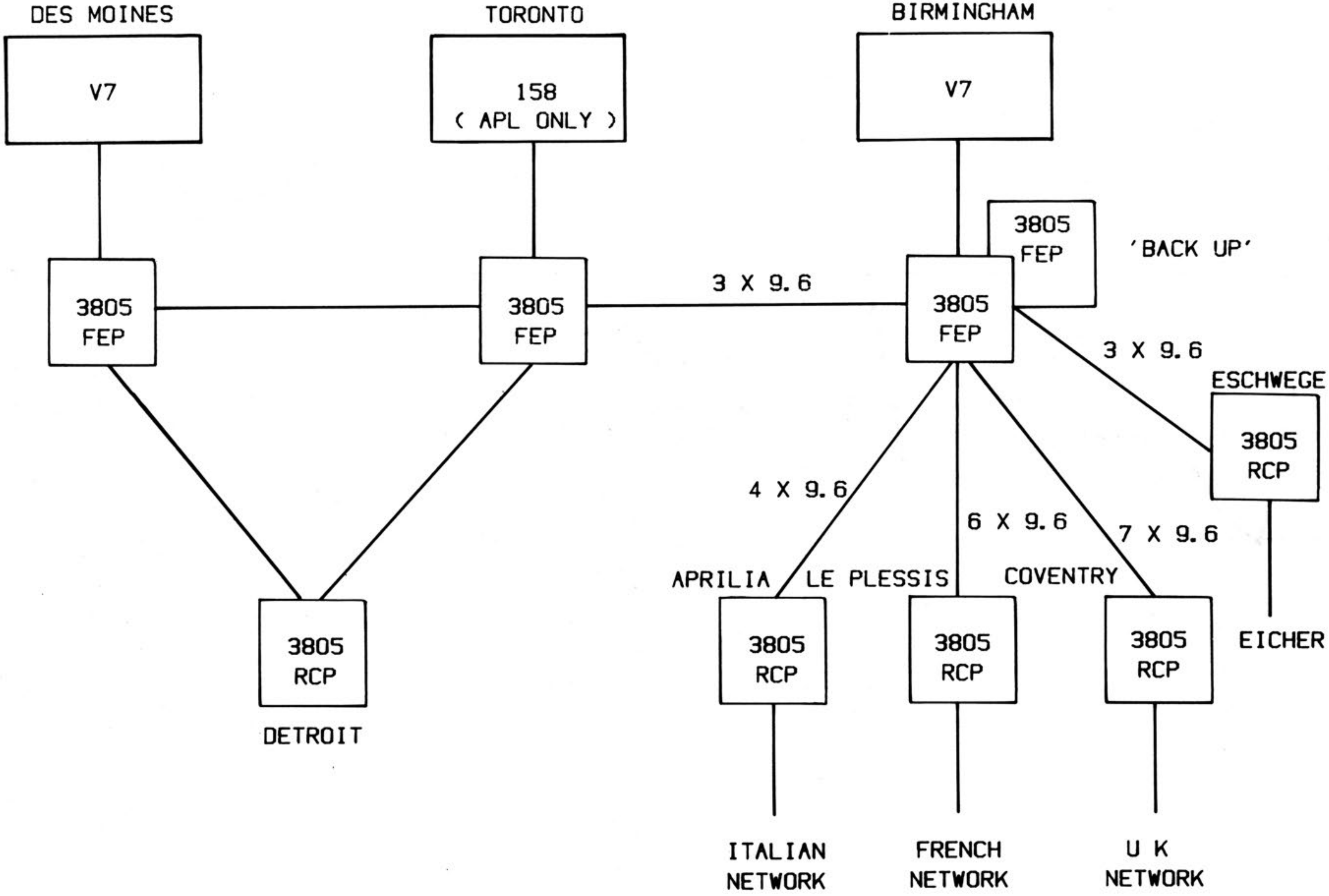
WORLD-WIDE MESSAGE COMMUNICATIONS NETWORK APRIL 1976



1978 TIME SHARING AND TELEGRAPH NETWORK



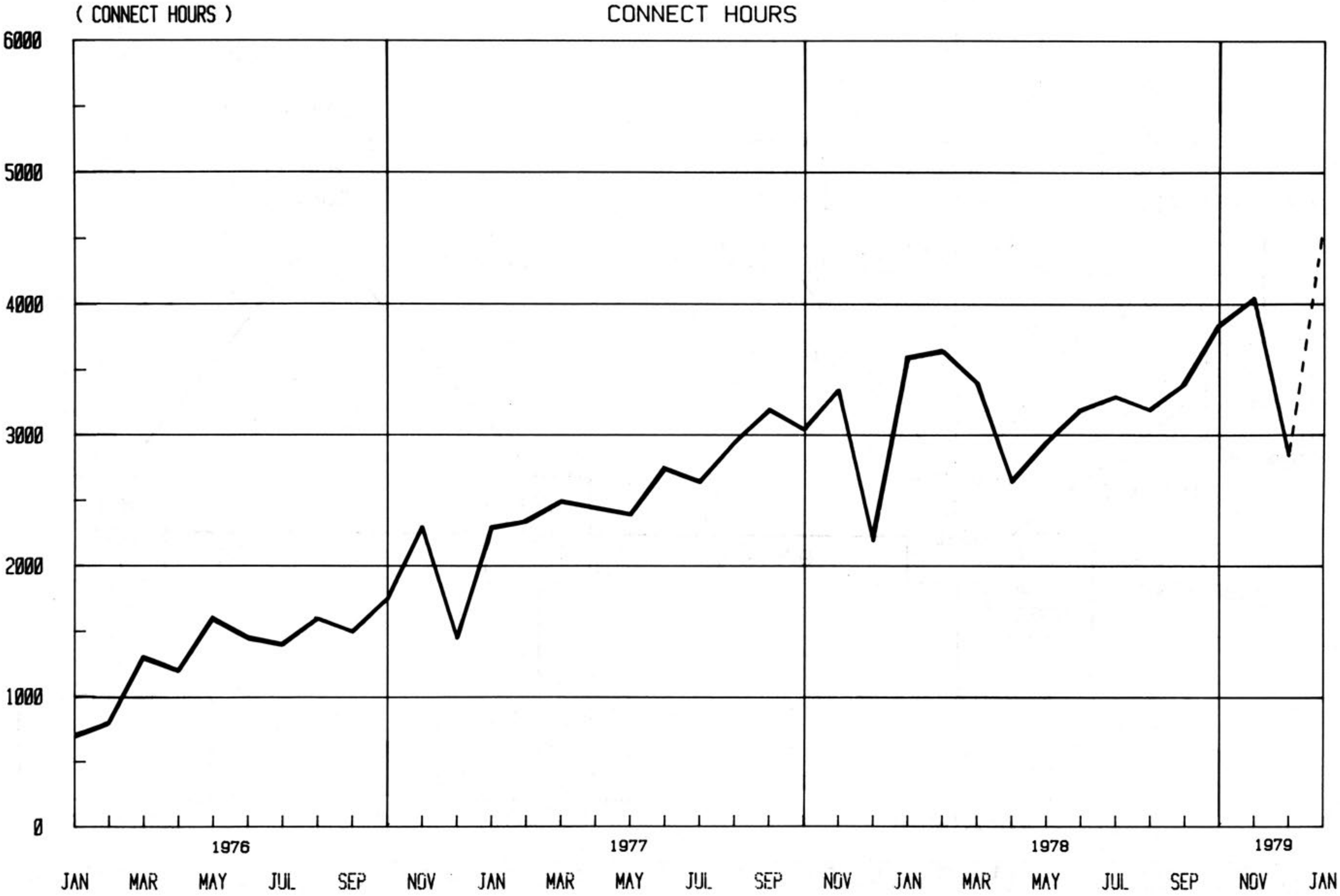
PROPOSED MASSEY FERGUSON 3805 NETWORK AS AT END OF 1980



INTERACTIVE COMPUTING USAGE

WORLD-WIDE GROWTH

CONNECT HOURS



AN INTERACTIVE AID TO PLANNING IN THE FOOD INDUSTRY

E.W. Burnside
Reckitt & Colman Limited
London, England

1. Introduction

This case history describes the development of an interactive planning computer model to aid in the development of strategies and tactics for the purchasing, production and blending of honey. The approach adopted by the project team was to use APL to quickly write prototype systems and to use these systems to help both the user and the project team develop an understanding of the user's problems and the interactions caused by these problems in other departments. During the development stages of the project emphasis was placed on the following:

- providing a computational aid to create thinking time for the user.

- not to take the decision making away from the user, but to provide more information to enable better decisions to be made.

- to provide a means of communication between user departments where all assumptions have to be explicitly stated.

- to provide accurate and up-to-date information on stock levels, expected deliveries, etc., to all user departments.

This paper records how the original brief of solving a short term blending problem was extended by phases into an integrated system to aid in the development of a long term purchasing strategy. The system which finally evolved is given detailing how the user, at run time, can dynamically redefine the criteria used to evaluate differing strategies and tactics. A brief description of how the system operates is also given.

Whilst this paper concentrates on the particular problem of honey blending, it is believed that the method used to design and develop the current system could have major advantages in the development of other systems. Some of these advantages are listed in the final section of this paper.

2. Background

Honey is a product produced by bees from the pollen of flowers. Some honey is sold in the cones in which the bee produces it, but, more usually it is sold in liquid form in one of two states — clear honey, which as its name implies is transparent, and set honey which is opaque. In the U.K., demand for honey greatly outweighs the supply, thus large quantities need to be imported. These honeys are from many countries including Australia, China, Rumania and Mexico. To ensure consistency of quality and taste these "raw" or source honeys need to be blended to give the final clear and set honey which are consumed at breakfast tables all over the country.

3. The Problem

The production of honey involves, in the main, five departments — Production, Purchasing, Formulation, Marketing and Finance.

The Marketing Department provides the sales forecast from which the production schedule is calculated within the inventory level laid down by the Financial Department. The Purchasing Department then draws up a purchasing plan within the financial constraints imposed by the raw materials inventory levels, and with regard to the possible blends as supplied by the Formulation Department (fig. 1). Traditionally the blends used to make the final clear and set honeys followed a well established recipe, which in times of difficulty in obtaining supplies of the required quality was replaced by a second choice or third choice recipe. This approach assumes that the quality of honey from a given location does not vary over time, and that supplies from traditional sources can always be obtained.

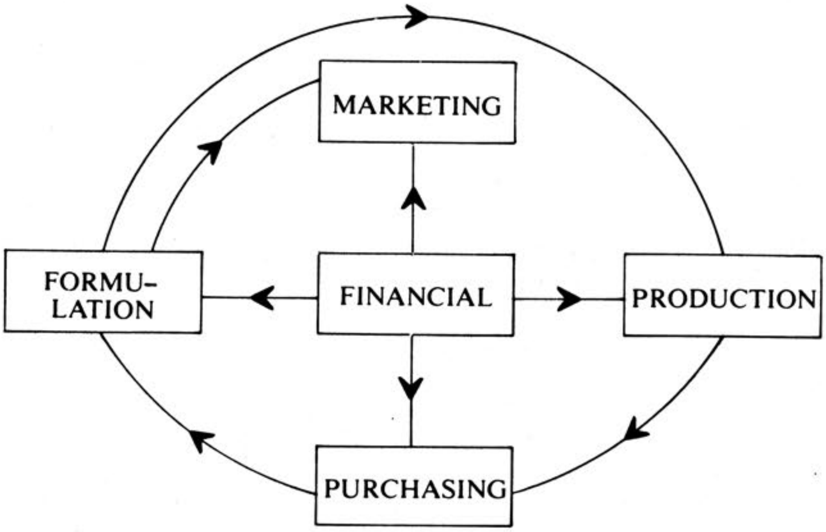


Figure 1

During 1978, a failure of the Australian honey crop (Australia is one of the world's largest exporters of honey) coupled with an increase in world demand, led to severe difficulties in obtaining raw honeys of the required quality. The recipe approach to blending had to be abandoned as the traditional raw honeys could not be obtained. Honeys of unknown quality had to be purchased from hitherto unused sources, with the quality parameters only being established after the shipment had arrived. Frequent reblending therefore had to take place, as continuity and quality of supplies could not be guaranteed.

The co-ordination of honey production was carried out by a Honey Working Party which had representatives from all the five departments involved. This working party, consisting of about 15 people, met frequently to consider the changing situation and to revise the production blend to be used as information on the quality of the most recent shipment of raw honey became available. The computational complexity of calculating a new blend was very time consuming, hence the first blend which satisfactorily met the quality requirements was accepted.

The Operational Research Department was called in by the R & D Director (to whom the Formulation Department reported to) to provide a computational aid to the blending of honey.

4. Development of the System

Phase I: Acceptance of a Computerized Approach

From initial discussion with the Formulation Department it quickly became apparent that a computerized approach was viewed with something less than overwhelming enthusiasm. In fact, computer systems were considered to be big, very complex, difficult to use, could not solve their problem, took a long time to develop, threatened their jobs, and would generally create a mess which they had to clear up. The first task of the O.R. person was to gain the confidence of the Formulation Department. This was achieved by quickly writing a simple program in APL to calculate the effect on the stock levels of any production schedule using a given blend for the clear and set honeys. This was an exercise which the Formulation Department performed every time there was a change in production schedule or blend, which during 1978 was very frequent. The stock level program was designed to reproduce the manual calculation. The output exactly duplicated the then current stock level documentation.

The program was written, tested, and being trialled by the Formulation Department, within a week of discussing the problem by which time they had, fortunately, the need to produce a new stock level plan. The speed of the computer system response and the ability to produce exactly what they currently did manually greatly impressed them. Indeed within a few days they were requesting extensions to the system. At all times it was stressed that the system written was only a prototype system and would be modified to meet the blenders' requirements.

Because the Formulation Department saw that modifications were quickly and easily carried out, they had no inhibitions in requesting them. The output from the stock level program quickly replaced the original stock level report.

Phase II: Development of a Blending Model

Phase I was very successful in gaining the Formulation Department's confidence in the ability of computer systems to help them in their work. They now became positively enthusiastic to try and extend the approach to solve the original problem, i.e., to provide a computational aid to honey blending. Together, the O.R. person and a person from the Formulation Department formed a small project team which defined the constraints worked to when preparing an acceptable honey blend. These constraints were as follows:

(1) Mass Balance constraints - that the production of clear and set honey should not exceed the opening stocks and the expected deliveries of raw honeys.

(2) Legal constraints - U.K. legislation specifies the minimum level of certain

quality parameters which must be met. These are the soluble solid content, the reduced sugars, the sucrose level, the hydrox F and the diastase number.

(3) Shelf life requirements - to ensure the shelf life of the set and clear honeys are acceptable, the L/D ratio must be above minimum level. This level differs for clear and for set honey.

(4) Consumer acceptability requirements - two constraints are placed by consumer acceptability requirements, the taste and the colour. Discussion established that the taste of the raw honeys and blended honeys could be graded on a 1-5 scale. Furthermore, the percentage of certain raw honeys from specified origins could not be exceeded in the final blend.

The primary objective is to meet all the above constraints using only raw honeys - any other additions being illegal. The second stage of choosing a single blend from the many possible satisfactory blends was, because of the computational difficulties not often encountered, the first satisfactory blend usually being accepted. However, the blenders expressed the desire to be able to test the effect of using differing criteria in choosing the "best" satisfactory blend, e.g., maximum colour, minimising solids, etc.

It was established that all the above constraints were linear so a Linear Programming approach was used. In order to give the blenders the facility to test the effect of changing the criteria of selection of the "best" blend, it was decided to write the L.P. model so that it could be used interactively, the main advantage being that many blends could be reformulated within a very short time.

The computer program giving the blenders all the above facilities was written in APL using an I.P. Sharp linear programming package available on workspace 544 *LINPROG*. Development of this system was completed in 2 - 3 weeks.

Phase III: Trialling of the System

At all stages in the contact between the Formulation Department and the Project team it was stressed that the systems being written were prototype systems, written by non-professional programmers, and that these systems would be professionally re-written when the user was more confident on the specification of the final system. User testing of the prototype systems was essential and criticisms and suggested modifications positively welcomed.

The Formulation Department, therefore, were requested to test the systems over a two month period, for all possible combinations of events. Initial training on the use of the system was given, and background support, if required.

During this two month period the system was given a complete testing in a live environment, sometimes to destruction. The only modification made to the programs were those that were essential to the running of the system, e.g., to overcome domain errors. At the end of the test period a long list of modifications were suggested mainly concerned with the ease of running the system rather than the logic of the formulation. The overall conclusion of the Formulation Department was that the system could calculate reasonable blends, and was a major advance in reducing their time involvement, but could be made easier to use.

All the modifications suggested by the Formulation Department were written into the prototype system.

Phase IV: Extension to Other Departments

Section 3 described how the production of honey, in the main, involves five departments: Production, Purchasing, Formulation Department, Marketing and Finance. The project team so far, had worked only with the Formulation Department. With their approval it was decided to extend the project brief to involve all the departments using the system and to provide an aid in the solution of the longer term planning problems as well as the shorter term blending problem. This was necessary to include the constraints imposed on the Formulation Department by these other Departments. To this end a presentation was made to the Honey Working Party, where suggestions were made on how the system could be extended to answer the question of what honeys to purchase in addition to the question of what to do with that honey when it has been purchased (i.e., how to blend it).

The proposal to extend the system was accepted and a member of the Purchasing Department co-opted onto the project team. To answer his purchasing questions of:

- quality of raw honey to purchase
- price to quote for raw honey
- contingency plans taking into account uncertainties on availability, quality, delivery, reliability, price, etc.

the model was extended to include the price of the blend as a constraint and also as a parameter in the selection of an acceptable blend.

Phase V: Trialling of Modified System

The Purchasing Department was requested to trial the modified system for a period of three months. Once again, it was emphasised that the system was still in prototype form and requests for modifications would be welcomed.

Phase VI: Re-writing the System

With both trialling phases succesfully completed, the Programming Department was called in to rewrite the system, giving all the facilities specified by the users but to professional standards, paying particular attention to the user/system interface, efficiency of coding, ease of maintenance and the Company's documentation standards. The users still had the prototype system to operate during this rewriting phase. When the rewriting was complete (2 months) full training of the user followed. The Programming Department retains the responsibility for maintaining the system.

5. System Description

As stated above, the total system consisted of two sections:

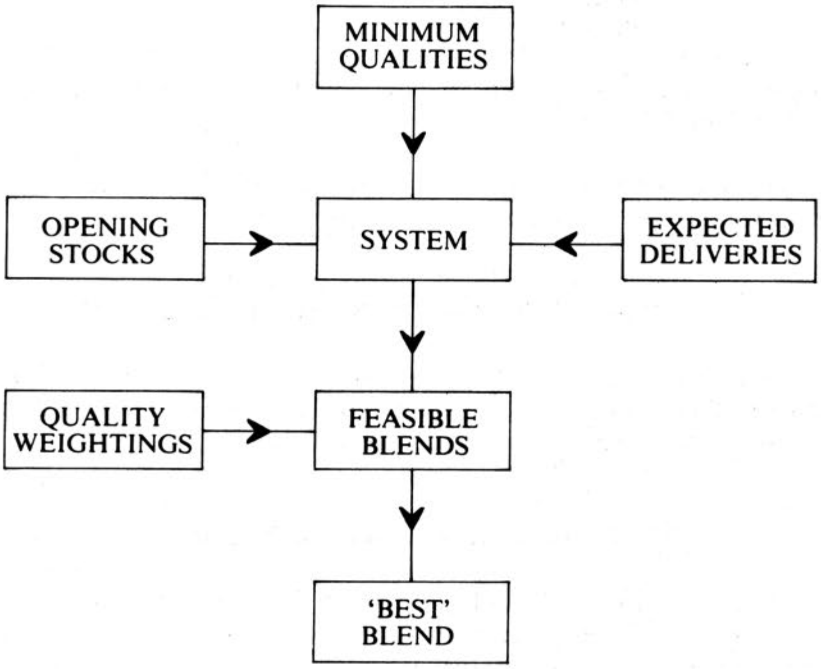
(1) Stock level calculation

(2) Blending/planning model

The two sections were designed to be an integral part of a total system, both referencing the same data file and with the chosen blend being able to be passed from the blending program into the stock level program. To facilitate the understanding of the system, the blending/planning model will be described first, even though it was written second, as the output from this module can be fed into the stock level program to check that adequate stock cover is available throughout the whole of the planning period.

Blending/Planning Model

Schematically, the system can be reviewed as follows:



The program plans for a single period which can be as long or as short as the user wishes (it is usually set at one year).

The system requires the following information which can either be via the keyboard or taken from a data file:

(1) Opening stocks of raw honeys

(2) Expected deliveries of raw honeys

(3) Quality parameters of these honeys

(4) Minimum permissible closing stock level for each raw honey

(5) The maximum percentage of each raw honey in the final clear and set formulations

(6) The production required for clear and set honey

(7) The weighting to be given to each clear and set quality parameter (including cost) when selecting the best blend from the set of feasible blends.

Because the system is written in APL it could become very expensive to run, the expense being geometrically in proportion to the size of the problem. To reduce the size, and hence the cost to a minimum, the facility was built into the system to enable the user to specify the quality constraints which must be met. Thus, if the user believes that, say, the reduced sugar content will always be above the required quality, that quality constraint can be left out of the formulation thus reducing the size of the problem. The value of this quality parameter will still be calculated and can be used in the selection of the "best" blend.

The first calculation the system makes is to check that there is enough honey available (stocks plus expected deliveries less minimum required closing stock) to meet the production requirements. If production requirements are greater than expected availability the user can choose between four options:

(1) Make the requirement equal to the availability

(2) Make the availability equal to the requirement

(3) Allow system to suggest blends and deliveries

(4) Cancel the run.

The system will calculate the "best" blend displaying

(1) Closing stocks

(2) Quality parameters not met (if any)

(3) Blend for set and clear

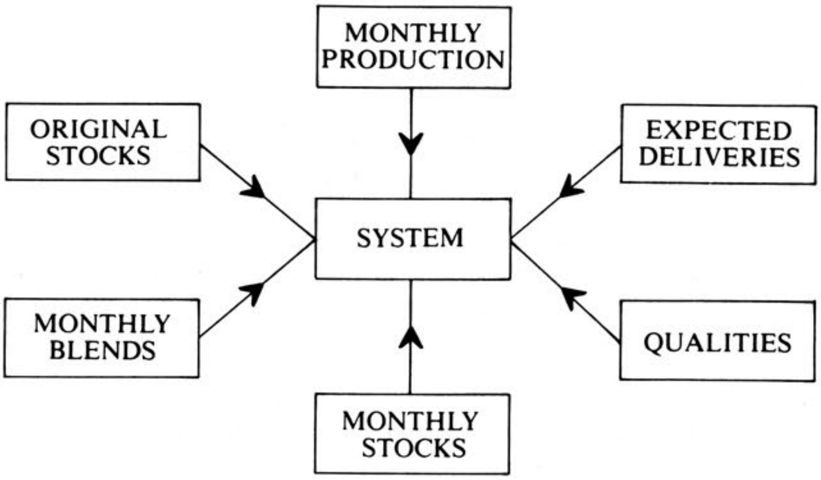
(4) Quality achieved

(5) Suggested deliveries, if requested.

The user then has the option of changing any of the current set-up values and re-running. Output can also be stored on a file.

Stock Level Calculation

Schematically:



Given the specified blending programme, the system calculates the monthly movement in stock. The blending programme can either be that calculated by the Blending/ Planning model or any other specified at run time. This report is required because it checks that the yearly schedule as calculated by the blending/planning model is feasible on a monthly basis. The same method of operation applies to both sections of the system insofar as the user can change any of the data and re-run at anytime.

6. Summary

The case history given in this paper records the development of a blending/planning model using a prototype system approach. This summary, therefore, will be divided into two sections, the first of which will describe the benefits obtained in the construction and use of the planning model and the second section will list some of the perceived advantages of using a prototype system approach.

Benefits of the Planning Model

The ultimate criteria on which to judge any system must be the acceptability of the system to the user and the use he makes of it.

The planning system described in this paper is used on a regular and frequent basis by the Formulation Department to calculate blends and by the Purchasing Department to evaluate planning strategies. The output from both the planning/blending module and the stock level calculation module have now replaced the manually prepared reports. The Honey Working Party now meets very infrequently, all departments being prepared to accept the calculations made by the system. In financial terms, stock levels of raw honeys have been reduced, because by better blending more of the source honeys can be used. The system also leads to a more effective use of the new honeys.

One of the major benefits of the system, however, is unquantifiable, that is the understanding gained of the total blending problem and the interactions between the different departments involved. The user now has to specify the criteria for the selection of the "best" blend and therefore has to define and specify "best". This aids in the communication between departments as all assumptions have now to be stated explicitly.

Last, but not least, is that the computer is seen as being apolitical, and therefore its calcualations are more readily accepted by all the user departments.

Benefits of a Prototype System Approach

The main benefit of a prototype system approach is that the user develops and modifies the system himself and therefore identifies closely with it. His understanding of his problems will evolve during development, and the system can quickly and easily be modified to accomodate these changes in understanding. At the end of the development phase the user is left with a system with which he identifies, which more closely meets his needs, and which he can more readily adapt to meet changing circumstances. A second major benefit is that throughout the development phase the designer of the system and the user are in close contact, the development phase usually being of a short duration. The problem owner therefore, never looses touch with the development of the solution, and hence will not solve this problem in another way. If the problem is immediate and urgent the user cannot wait up to a year which can be the usual time scale for the development of a computer system by conventional methods. The final system may be the same, but throughout the "professional" writing phase the user will have his prototype system to use.

7. Conclusion

It is firmly believed that in the development of this planning aid the prototype system approach offered major advantages over conventional computer development methods. The approach aided design, user acceptance of this system in particular and computer systems in general, communications between departments and in the final analysis, the attainment of the financial benefits of stock reduction the system could generate. The use of APL was fundamental in this approach, because it was only by the use of an interactive interpretive language that the flexible design of resizing the problem and the redefinition of the objective function at run time could be used. APL also aided in the speed of development and modification.

THE GROWTH AND FUTURE OF APL AT FIREMAN'S FUND

Eugene R. Mannacio
Fireman's Fund American Life
San Rafael, California

Introduction of APL to Fireman's Fund

When I joined the Actuarial Department of Fireman's Fund in May, 1974, none of us had ever used APL. Our actuaries were using FORTRAN and PL/1 on TSO. Response time was terrible, on-line debugging features were non-existent and, since we were not "programmers" by profession, our programs were relegated to run on the "test" system rather than the "production" system. Despite these problems, when a newly hired actuary suggested we use APL, we were skeptical. But we were convinced when we saw the dramatic increase in our own level of productivity using APL.

Of course, it is not unusual that actuaries should be doing computer programming of their own. The highly complex and technical calculations an actuary must do, requires that he be given a free hand to do the programming. It is not surprising, therefore, that APL should be very popular with actuaries.

Rapid Growth of Its Usage - What and Why

We were introduced to APL through time-sharing with STSC. Because we found APL such a powerful problem solving tool, our usage grew very rapidly. Figure 1 shows how our monthly usage has increased. Of course, a rapid growth in usage implies a similar growth in expenses. Expenses that grow at this rate become a matter for close scrutiny in our company. Corporate executives expect to be able to budget for growth. What applications are we developing and running, and why do we use APL? These are questions we are still being asked.

To some extent, an actuarial department can use the technical nature of the work it does to justify a language that the company considers non-standard. If it seems to be more productive and is making important contributions to the profit making ability of the company, it is unlikely ever to be in jeopardy of losing the APL facility. However, our own actuary was still interested in making sure that efficient use was being made of APL.

Our monthly bills would only tell him how much was spent on a particular account number, but not what it was spent for. His first suggestion toward controlling usage was to have a log. Each time a user signed off, he would have to put his name on the log, the project he was working on, and how much he spent during that session.

Tools That Were Developed to Monitor Usage

Had this suggestion been adopted, it would have added an unwelcome ritual to the sign-off procedure. Moreover, it might have discouraged a user from having many short terminal sessions instead of one long session, since it would be necessary to manually calculate his cost for each session.

```
                          APL USAGE AT FIREMAN'S FUND


         CONNECT            STORAGE                   CONNECT           STORAGE
MONTH    HRS:MIN     CPU      MBD            MONTH    HRS:MIN     CPU      MBD
-----    -------     ---    -------          -----    -------     ---    -------

 1/75     09:56      145          7          10/77     288:04    9617        575
 2/75     14:49      181         12          11/77     321:19   10472        637
 3/75     28:45      172         10          12/77     200:20    6311        658
 4/75     15:19      323         17
 5/75     08:12       49         17           1/78     241:50    9329        654
 6/75     87:24      645         24           2/78     175:07    6220        776
 7/75    200:12     2495         45           3/78     329:25    8271        840
 8/75    108:26     1048         66           4/78     258:07   10759       1001
 9/75    140:11     1688        112           5/78     221:55    9442       1122
10/75    163:43     2264        116           6/78     336:39    9394       1048
11/75    103:01     2597        129           7/78     244:27    8389        728
12/75    127:08     2281        139           8/78     261:58    8411        504
                                              9/78     262:47    9818        479
 1/76    135:49     6720        166          10/78     251:33    8733        508
 2/76    164:54     2338        162          11/78     214:07    8221        620
 3/76    179:46     4041        186          12/78     289:04   10801        671
 4/76    144:32     3095        220
 5/76    146:57     2784        197           1/79     289:44   13735        663
 6/76    175:00     6483        182           2/79     311:04    8018        550
 7/76    111:23     3785        213           3/79     324:39   12566        667
 8/76    107:21     4312        224           4/79     316:30   10376        593
 9/76    179:12     4641        243           5/79     220:16    7139        602
10/76    204:03     4145        291           6/79     270:34    9195        619
11/76    185:15     4767        301           7/79     343:02   12128        725
12/76    204:41    10704        371           8/79     261:43    9299        772
                                              9/79     220:43    7989        684
 1/77    196:16    11069        397          10/79     303:22   10998        739
 2/77    143:08     3617        376          11/79     330:34   14350        985
 3/77    198:52     4268        463          12/79     273:21   10874        952
 4/77    218:48     4124        496
 5/77    226:08     7157        573           1/80     292:05   12356        979
 6/77    256:29     4901        491           2/80     284:44   12101       1118
 7/77    240:33     6938        492           3/80     392:42   17676       1095
 8/77    265:33     9961        578           4/80     439:10   15921        828
 9/77    319:49    14174        750
```

Figure 1

Instead, I developed a program we have been using ever since called GOAWAY. This program automatically calculates the cost of the terminal session, puts the connect time, CPU time, user number and WSID at sign-off on the bottom of a matrix in a file component corresponding to the number of the month and performs the equivalent to *)OFF HOLD*. The assumption is made that the name of the workspace at sign-off will be mnemonic and help to identify the project.

```
    ∇ GOAWAY;OFF
[1]   ⎕FUNTIE ⎕FNUMS ⋄
      'NUMBER ACCOUNT' ⎕FSTIE 93 LOCK ⋄
      ⍎⎕DEF ⎕FREAD 93 13 LOCK
   ∇

      THE LOCAL FUNCTION <OFF> STORED IN COMPONENT 13 :

    ∇ OFF;X;Y;Z;W;M;⎕IO;⎕PW;CN;UN
[1]   ⎕IO←1 ⋄
      X←⎕NL 2 3 ⋄
      X←(~(((1⍴⍴X),3)↑X)∧.='ΔWS')⌿X ⋄
      ΔWS←12↑⍎((2=⎕NC 'ΔWS')/'ΔWS'),',¯12↑⎕WSID' ⋄
      ⎕PW←130
[2]   X←⎕EX X ⋄
      ⎕FHOLD 2 1 ⍴ 93 ⋄
      Z←6⌶11,0 ⋄
      UN←⎕TASKS[⎕TASKS[;1]⍳⎕TASK;4]
[3]   'YOUR COST FOR THIS SESSION IS: ',Z← 8 2 ⍕+/RATES×0.001× 280 3000 +⎕AI[2 3] ⋄
      CN←(14,⎕TS[2])[1+⎕WA>70000]
[4]   X←⎕FREAD 93,CN, ⋄
      →((CN=14)∨0=1⍴⍴M̲←⎕FREAD 93 14)⍴A̲ ⋄
      X←X,[1] M̲ ⋄
      (0 53 ⍴' ') ⎕FREPLACE 93 14 LOCK
[5]   A̲:M̲←(6 0 ⍕⌶25),((8 0 ⍕UN),ΔWS),' ',Z,' ', 8 0 8 0 ⍕⎕AI[2 3] ⋄
      →(14=CN)⍴B ⋄
      →B×⍳(⎕FI X[1; 5 6])=⎕FI M[5 6] ⋄
      X← 0 53 ⍴' '
[6]   B̲:X←X[1] M̲ ⋄
      X ⎕FREPLACE 93,CN, ⋄
      ⎕FUNTIE 93
[7]   X←6⌶11,2 ⋄
      Z←6⌶11,0 ⋄
      →
    ∇
```

By requiring that everyone use GOAWAY to sign-off, costs can be conveniently tracked. A reporting program called BREAKDOWN allows the steward to get a complete list of CPU and connect costs aggregated by workspace within user number and subtotalled by user number. To the extent that this listing does not agree with our actual billing, we know that either the user has not been signing off with GOAWAY or has had problems with line drops or crashes. The file organization chosen allows us to keep a running 12-month history of our time-sharing usage. If the number of terminal sessions per month significantly exceeds 1,000, workspace limitations will require that we change this organization. Since the current version of GOAWAY does most of its work within a local function that is read from a file, the programming change required could be accomplished without impacting its users.

In addition to this monitoring tool we have also made use of a proprietary package of STSC's called the Computer Usage Data System, or CUDS, for short. By requiring that ongoing systems be put on separate account numbers, we can produce a statement like the one illustrated in Figure 2 even before we have received our bill from STSC. In addition, since APL has grown beyond the Actuarial Department (as will be explained in the next section), this statement also allocates the expenses to the departments in which they were incurred.

APL Applications at the Fund

Some of the first APL applications we developed were systems to do:

A & H Reserves
Group Annuity Reserves
Monthly Budget Tracking
Pension Supplemention Reserves (for LTD)
Asset Share Calculations

At the time these applications were developed, the cost of ongoing systems was only 40% of our total monthly billings. The other 60% was spent for "one-time" projects - doing rate developments for new products, re-rating existing products, etc.

When APL is used primarily for "one-time" projects, the design and documentation standards can be fairly lenient. To the extent that APL was being used only as a computational tool by the Actuarial Department, this leniency did not create any problems. In fact, it seemed to enhance the claims we made about how fast we could get things done in APL.

We first discovered the pitfalls of writing full-fledged systems in this manner in December, 1975 (less than a year after we signed up with STSC). Given only two weeks, I wrote a group annuity valuation system by making modifications to an existing but much more limited routine. It worked, and we got our reserves done, but it used a lot of CPU cycles. In fact, it cost us over $5,000 just to do reserving on a few thousand people. I immediately suggested the system should be rewritten, but it was almost two years before this was finally done (although a rewrite would result in savings of $4,500 per year).

Standards for Writing Systems

In order to make sure we didn't repeat these mistakes, I began establishing some standards and procedures for systems writing. In addition, I developed some utilities which have become the accepted standard.

One example is an especially refined AKI program. Almost anyone who has written a few systems has an AKI program (Accept Keyboard Input). But nowhere had I seen the entire question of accepting input thoroughly addressed. Each time input is accepted, the programmer usually wishes to: issue a prompt, check the validity of the data and, if invalid data is entered, issue an error message, with the user having various options, including reentering the data, editing it or escaping. Most implementations separated these steps into modules so that the programmer still had to code conditional branches in the higher level routine over and over again. The utility I wrote uses two files; one has a single prompt in each component, the other has a matrix of error messages in a component corresponding to the prompt. Its left argument is the component number of the prompt (or error message matrix), its right argument is the quoted name of the validation routine. The validation routine takes as its right argument the data given in quote-quad input to AKI. If valid, it returns the data; if invalid, it returns an empty array whose first dimension is 0 and whose other dimensions are the numbers of the tests that were failed. If AKI sees valid data returned, it in turn passes the data to the driver as an explicit result; otherwise, it gives the user the option to reenter, edit (using a CH type function), or continue later (be signed off automatically with a)*CONTINUE*). One slightly non-standard aspect of this entry routine is that it takes field oriented entries so that the validation can be done on all records at the same time. Using this routine, other APL programmers in our company have begun to find systems writing easier.

Standardizing our approach to systems writing is only a first step. It is also necessary to ensure that those who program in APL are more thoroughly trained in its proper use. Although APL is friendly to the inexperienced user, it can easily be abused. Learning efficient coding practices in APL is probably more difficult than learning good style in some other languages. Also, good APL instruction is hard to find. Accordingly, we have begun to establish our own training and testing procedures. When these procedures are fully established, we hope that they will provide the structure and discipline that are conducive to good APL programming.

But right now, we have more than 20 running systems which by themselves have produced substantial growth in our expenses. We can no longer justify these expenses by saying APL is a computational tool for the actuaries. And we don't want to face the possibility that a high level corporate decision might eventually force us to severely restrict, if not curtail, our usage entirely.

Accordingly, we have been exploring alternatives to time-sharing with STSC.

Exploring Alternatives to Time-sharing

Our first explorations into alternatives were made as early as January of 1976. Some of our basic requirements were:

1) An APL component file system,
2) A commercial formatter,
3) Reasonable response time.

These requirements didn't seem too stringent. In our view, using APL without a component file system is completely impractical in a business environment. How else can one conveniently manipulate a data base which may be many times as large as a workspace? Without a commercial formatter, reports are simply too cumbersome - thorn, is not, by itself, sufficient. And, of course, why bother using a language which improves the efficiency of programming through its interactive facilities if the response time is so poor it obviates all of those advantages?

There were several options we could investigate, but one of the most obvious to consider was to bring APL in-house on one of our large mainframes. This would have the advantage of utilizing already existing facilities and, therefore, being lowest in cost.

Unfortunately, IBM didn't have a component file system and time-sharing companies were generally reluctant to sell us their software.

Purchasing a mini-computer was another possiblity. But APL implementations on minis were in their infancy. The implementations, as such, all suffered from some important deficiencies, such as small workspaces or lack of component files. The HP3000 had one of the most interesting implementations. Its features included virtual workspaces, a component file system, a commercial formatter, immediate execution editing and error trapping. Unfortunately, it did not provide the response time we were looking for.

In the fall of 1977, we started to try a second approach — to look for cheaper time-sharing rates. A company which had recently gone into APL time-sharing, offered us an attractive proposition: rates 30-40% lower than STSC. Like STSC, they were running on an Amdahl. They had just purchased a component file system which used SV and an auxiliary processor. When I ran some limited preliminary tests, there appeared to be no problems. So we began doing a conversion.

Unfortunately, the file system was not really as stable as it first seemed. In fact, it was brought down every time an APL emulation of a background file sort was run. We were promised these problems would be eliminated, but many deadlines passed without the corrections being made. So we returned to STSC.

Although this experience didn't involve any large direct expenses, we found that a conversion effort of this kind can involve a considerable cost in manpower. Accordingly, it is quite important to do everything possible to assure the conversion will be a success before attempting it.

At this time, we are still looking for in-house alternatives. Since we last did a study with the cooperation of our Computer Services Department, a number of things have changed, which makes us believe we may be able to find a viable in-house alternative. First, our monthly billings are continuing to rise (they are now in excess of $15,000 per month). Second, mini-computers are becoming less expensive (spurred, perhaps, by IBM's introduction of the 4331 and 4341). Third, new APL implementations have been introduced, such as HARRIS' APL (which, incidentally, includes an APL file system and is expected to include a commercial formatter very soon). Fourth, various time-sharing firms, including Sharp and STSC, have now begun selling their software for installation in-house. We believe that these developments make it possible for an APL user of our size to expect to be able to convert. Of course, it should be pointed out that we have been careful to avoid tying ourselves to proprietary packages. To us, it is APL which affords the greatest advantages in increased productivity, not pre-programmed packages.

APL Grows Beyond Actuarial

As the Life Company began to see what we could do, more and more APL applications were added. Other departments, too, began to have their own requests. It seemed to me that someone was needed specifically to handle these requests and manage our APL growth, so I proposed this. Our President approved of the concept and a year and a half later, the position was put in the budget. I have been in that position for over a year now and am beginning to see other departments recognize that APL could be doing more for our company.

Quo Vadis APL?

All of us who have used APL would like to see it used more widely. Not, I hope, because we have closed our minds to the possibility that some day there may be a better language, but, rather, because we have been able to be so productive in APL. In the commercial environment, however, I think we must admit APL has achieved only very limited success.

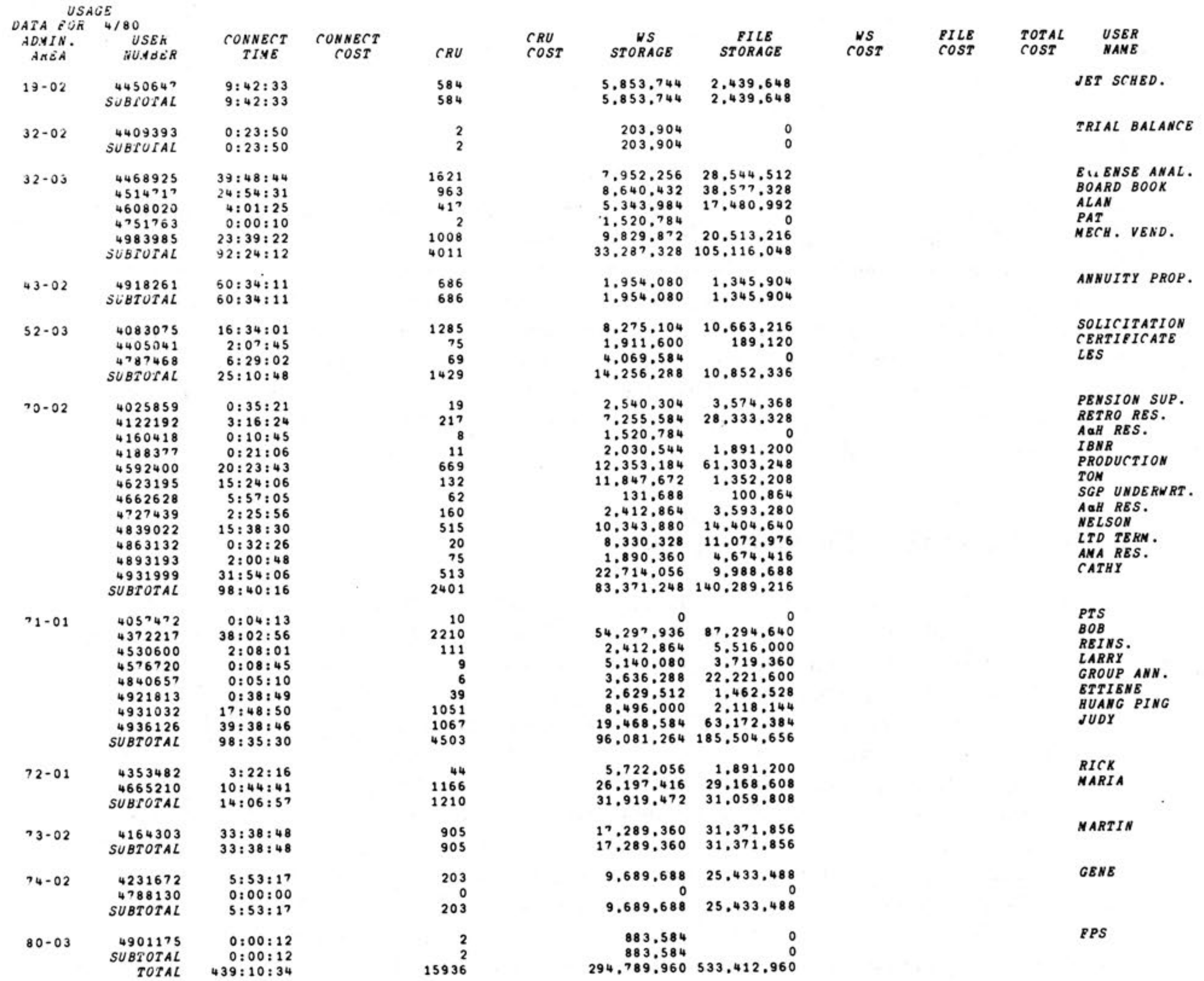
There are many reasons businesses have chosen not to use APL. Managers and programmers in business are sometimes put off by the symbolic nature of the language. Since it looks so abstract to them, they imagine it is more difficult to document. Programmers who have spent years acquiring proficiency in another language are reluctant to do it all over again in APL. Finally, APL has acquired a reputation for using much more computer time. But the reason businesses have chosen not to use APL are clearly overshadowed by the reason they should: APL is very cost effective. Properly trained, an APL programmer can produce far more than his counterparts in other languages. A well trained programmer will also provide adequate documentation so that systems can be efficiently maintained. Although the computer time an APL system uses may be somewhat larger than the time that would be used by other languages, this is, in most cases, more than offset by the increased programmer productivity. Unfortunately, business has been slow to appreciate this advantage.

Certainly, it would be impractical to expect the immediate replacement of the many COBOL programs that have been written, with APL programs. And, for large processing applications, it may well be true that COBOL's file processing is more efficient **now**.

But even in the area of management reporting and computing done at the user level, we have not even scratched the surface. Many companies, like my own, are now going into distributive data processing in a big way. APL could be the ideal vehicle for distributive systems, but, to my knowledge, no one has set up a distributive system based in APL. One important reason for this is a lack of education. Not enough programmers are being acquainted with APL and not enough managers are being shown how much more effective it can make their staff. To date, the studies showing how much more efficient the APL programmer can be are few and inconclusive. Some APL timesharing firms have taken the view that it is more profitable to write packages for customers in APL than to show them the benefits of using APL themselves. To my way of thinking, this situation must be changed if we ever hope to make a real impact on the data processing community.

Within my own company, I have outlined the steps that have been taken to help make APL programming more cost effective. As these measures show results, I expect APL to continue to grow as the Life Company makes more demands of the quicker turnaround and greater flexibility APL systems can provide. But I would hope that the growth of APL does not lead to a centralized APL programming resource with all the attendant problems. One of the problems of a centralized resource is its isolation, which has detrimental effect on communications. And when communications between a D.P. department and the user are not good, then the user's programming requests ultimately result in systems which do not fully address his needs.

Our actuarial Department develops systems more rapidly and with greater user satisfaction not only because it programs in APL, but also because it does its own programming. The people assigned to do the job always have direct experience with the problems they are to solve. They understand the facts and consequences that could easily be missed if the problem were put on paper. They may not all be trained on the most efficient ways to use APL, but this can be changed. To my way of thinking, more programming should be done in this way by people assigned to work within the departments that have the need (people who are expected to be specialists in two areas, their department's needs and their computer language — APL, of course). In such an environment, I believe we can expect the higher level of acceptance and achievement I know we all look forward to.



USAGE
DATA FOR 4/80

ADMIN. AREA	USER NUMBER	CONNECT TIME	CONNECT COST	CRU	CRU COST	WS STORAGE	FILE STORAGE	WS COST	FILE COST	TOTAL COST	USER NAME
19-02	4450647	9:42:33		584		5,853,744	2,439,648				JET SCHED.
	SUBTOTAL	9:42:33		584		5,853,744	2,439,648				
32-02	4409393	0:23:50		2		203,904	0				TRIAL BALANCE
	SUBTOTAL	0:23:50		2		203,904	0				
32-03	4468925	39:48:44		1621		7,952,256	28,544,512				EXPENSE ANAL.
	4514717	24:54:31		963		8,640,432	38,577,328				BOARD BOOK
	4608020	4:01:25		417		5,343,984	17,480,992				ALAN
	4751763	0:00:10		2		1,520,784	0				PAT
	4983985	23:39:22		1008		9,829,872	20,513,216				MECH. VEND.
	SUBTOTAL	92:24:12		4011		33,287,328	105,116,048				
43-02	4918261	60:34:11		686		1,954,080	1,345,904				ANNUITY PROP.
	SUBTOTAL	60:34:11		686		1,954,080	1,345,904				
52-03	4083075	16:34:01		1285		8,275,104	10,663,216				SOLICITATION
	4405041	2:07:45		75		1,911,600	189,120				CERTIFICATE
	4787468	6:29:02		69		4,069,584	0				LES
	SUBTOTAL	25:10:48		1429		14,256,288	10,852,336				
70-02	4025859	0:35:21		19		2,540,304	3,574,368				PENSION SUP.
	4122192	3:16:24		217		7,255,584	28,333,328				RETRO RES.
	4160418	0:10:45		8		1,520,784	0				AaH RES.
	4188377	0:21:06		11		2,030,544	1,891,200				IBNR
	4592400	20:23:43		669		12,353,184	61,303,248				PRODUCTION
	4623195	15:24:06		132		11,847,672	1,352,208				TOM
	4662628	5:57:05		62		131,688	100,864				SGP UNDERWRT.
	4727439	2:25:56		160		2,412,864	3,593,280				AaH RES.
	4839022	15:38:30		515		10,343,880	14,404,640				NELSON
	4863132	0:32:26		20		8,330,328	11,072,976				LTD TERM.
	4893193	2:00:48		75		1,890,360	4,674,416				AMA RES.
	4931999	31:54:06		513		22,714,056	9,988,688				CATHY
	SUBTOTAL	98:40:16		2401		83,371,248	140,289,216				
71-01	4057472	0:04:13		10		0	0				PTS
	4372217	38:02:56		2210		54,297,936	87,294,640				BOB
	4530600	2:08:01		111		2,412,864	5,516,000				REINS.
	4576720	0:08:45		9		5,140,080	3,719,360				LARRY
	4840657	0:05:10		6		3,636,288	22,221,600				GROUP ANN.
	4921813	0:38:49		39		2,629,512	1,462,528				ETTIENE
	4931032	17:48:50		1051		8,496,000	2,118,144				HUANG PING
	4936126	39:38:46		1067		19,468,584	63,172,384				JUDY
	SUBTOTAL	98:35:30		4503		96,081,264	185,504,656				
72-01	4353482	3:22:16		44		5,722,056	1,891,200				RICK
	4665210	10:44:41		1166		26,197,416	29,168,608				MARIA
	SUBTOTAL	14:06:57		1210		31,919,472	31,059,808				
73-02	4164303	33:38:48		905		17,289,360	31,371,856				MARTIN
	SUBTOTAL	33:38:48		905		17,289,360	31,371,856				
74-02	4231672	5:53:17		203		9,689,688	25,433,488				GENE
	4788130	0:00:00		0		0	0				
	SUBTOTAL	5:53:17		203		9,689,688	25,433,488				
80-03	4901175	0:00:12		2		883,584	0				FPS
	SUBTOTAL	0:00:12		2		883,584	0				
	TOTAL	439:10:34		15936		294,789,960	533,412,960				

Figure 2

A PERSONNEL RECORDS AND INFORMATION SYSTEM

Chris Baron
BL Cars Limited
Coventry, England

Introduction

This paper describes the development and implementation of a personnel system within BL Cars. The description is not technical, since no internal APL expertise was used. It relates how, in a company of 100,000 employees and 35 main sites, a system can be set up quickly, on a timesharing bureau, to meet business needs. It discusses the problems and benefits of establishing a major system with local, terminal-based input and output, operated by personnel clerks. It considers the extent to which local autonomy and central control are desirable, the effectiveness of the system, and how it has been developed to meet specific local and central needs. It examines what further developments are desirable.

In The Beginning ... The Working Party

The story of personnel systems within BL Cars goes back a long time, but for this paper we will take the start date as 1974. In that year a BL Limited working party produced a paper which identified certain basic needs: that there should be a common approach to payroll systems, and a common approach to personnel systems, and that these should be linked. This philosophy has continued unchanged — but that does not mean that we have yet achieved the objective. BL had been formed by gradual amalgamation of many, many different firms, and over the years very little had been done to standardise any aspect of personnel or payroll. The task ahead was horrendous.

In 1975, the company undertook a major re-organisation, which resulted in the setting up of Cars as a major (but, in business terms, largely distinct) part of BL. Fairly soon a Cars working party was established to develop common personnel policies and procedures, and personnel systems was one of the many agenda items. It was a large working party, with representatives from all levels of the organisation (company, division and plant) and relevant functions (including Finance and Systems). It served a useful purpose in a number of ways, particularly as a consultation and communication medium, but suffered somewhat from the normal problems of large working parties — lack of direction, difficulty in decision making, and long discussions about relative trivia.

By 1977, the company was moving steadily, if slowly, toward the adoption of a common payroll system; this in itself was a major undertaking, due to many difficulties created by government pay policies, changes in payment structures (including the implementation of flat rate payment for production workers and their associated lay-off payment schemes), organisational complexities and last (but by no means least) the fact that we had nearly 120 different payrolls. Add to that, of course, the traditional shortage of competent computer people.

On the personnel side, BL Cars was moving towards common payment and job grading systems, and the introduction of common sick pay schemes and other terms and conditions. Since these needed to be developed and implemented (and would subsequently need to be maintained and be open for re-negotiation), and since there were also many other personnel problems which might be eased by the availability of better personnel information, it was natural that the working party had, during 1976, turned its attention more towards personnel systems.

During 1977, I became involved through my manpower planning interests, and by mid 1977 a proposal was produced showing our plans for an all singing and dancing system — pilot implementation, 1980. Top management said:

"Great, but we need something sooner"

The Gestation Period

It so happened that in early 1978 I had some specific manpower planning problems searching for computerised assistance (analysis of a certain category of workers that we were desperately short of, to see if things were getting better or worse and where we should direct our energies, and the planning of a major manpower transfer/redundancy exercise). Our systems people put us in touch with I.P. Sharp Associates (BL's "preferred", but not only, timesharing supplier) and Hugh Hyndman, who, by chance, had some systems ideas looking for an application. For various reasons totally unconnected with computers, these two jobs came to nothing, but we did spend a little time discussing what facilities we would be looking for in a personnel information system.

Thus, when the management said they wanted something sooner, a subcommittee was set up and I found myself on it. We spoke to other firms, and the Institute of Manpower Studies, and soon had a reasonable idea of what we needed out of a quick and dirty personnel system — it should be:

- integrated with payroll input and output;
- integrated with administration;
- flexible in outputs;
- flexible to meet local needs for storing data;
- easy to introduce quickly;
- easy to use by clerks;
- reliable;
- transparent to organisation changes;
- secure;
- available from any Company location or level;
- cheap;
- designed to allow appropriate local and central control.

These requirements put some restrictions on the system, most obviously:

- the data on each record should be severely limited;
- rapidly changing data should be avoided;
- standard data and codes should be used where possible;

- different data may be appropriate for different categories of employees;
- the data must be held in a central "bank";
- the involvement of our own (already stretched) systems people should be as limited as possible.

Our first idea was to run the system off payrolls onto timesharing, but there were some problems:

- too many payrolls;
- lack of standardisation (even between the "standard" payrolls);
- no common coding;
- too heavy a systems workload to change payrolls;
- additional complexities of negotiating with the Finance function who "own" the payrolls;
- lack of alignment between employees' locations and payrolls, and the administrative and organisational responsibilities;
- formal procedures for maintaining new data would be needed.

These all added up to give probable major implementation delays, so we were forced to reject the payroll base, and similarly found we would have big problems if we tried to develop an administration system as such from the start.

Timesharing, however, seemed to us necessary in order to achieve fast and flexible development and ready access for outputs, so we had to go for updating direct from the personnel departments. Our requirements (it has to be able to do everything and cost nothing) went out for quote, and in November 1977 we agreed that I.P. Sharp Associates should have the job, Hugh having meanwhile carried out further development work on the basis of our previous discussions. Although our systems people were confident that this decision was right, we, the users, were concerned: we knew that Systems' original choice of I.P. Sharp Associates as preferred timesharing supplier had been on the basis of totally different sorts of application (correct), and felt that I.P. Sharp Associates in the U.K. did not have the resources to develop and support what — even then — we saw as being a large, complex and demanding system (incorrect feeling, correct view).

Meanwhile, we had been looking hard at probable outputs. Our surveys included types and frequency of output requirements and perceived needs for historical data. In practice, this gave us a good guide to the data to be held, but little more, though had we had more time this information could have been put to better use.

We tried to define what fields to set up, taking account of need for data, ease of maintenance and cost. Most were obvious, but we were designing a system to serve sites varying in size from 150 employees to 25,000, so there were some decisions to take. Although the systems should be flexible, to hold different fields, we saw a danger of creating too complex a situation if we allowed too much flexibility. We did determine, though, that we would hold more data for the (5,000) corporate (management) staff than for the 15,000 sub-corporate (clerical, technical and supervisory) staff and in turn less for the 80,000 hourly paid employees. In particular, we aimed to hold more historical data for the corporate staff and none for the hourly.

With not the slightest idea of how the system would work, we also thought about field formats and coding. As far as possible, we tried to tie our coding in with that used by Finance or asked our most responsible potential users to design the structures. Neither way really worked — everyone tends to see things from their own viewpoint

— whether we were looking at largely factual items (just what is a payroll?) or semi-factual (e.g., organisation) or largely judgemental (e.g., activity coding). In the end, we mostly borrowed codes from other people or worked out our own.

Largely as a result of our thoughts on fields, we also decided upon the major organisation of the system as we saw it (Hugh, I am sure, might look at it differently). We would reflect the major personnel records responsibilities for hourly and sub-corporate employees, with one system for each group based on each of our 35 or so plants (70 systems in total); the corporate staff would all be covered by one system irrespective of their location or organisation. Leavers would be kept on the files, though may later be moved off onto special files if necessary.

We also decided that we would pilot the implementation at one plant. I wanted to go into the corporate staff area, because that was where I felt the biggest benefit lay, but there would obviously be difficulties created by the additional organisational and data complexity, so we decided on a plant. Unfortunately, we chose a plant of around 8,000 people — much too big for a pilot — but the groundwork had been done there in the earlier exercise and it would have been wrong if we had pulled out, having maintained our links up to then.

And then it all started.

The Birth of PRIS - The First System

In November 1977, we had a change of Managing Director, and by mid December it was obvious that a major re-organisation was coming. Re-organisations tend to have their most significant effect amongst the management, and we therefore proposed to the responsible director that we should set up the new Personnel Records and Information System (PRIS) for all corporate staff, to assist him in managing the re-organisation. He agreed.

By mid January we had pulled together all the strings we could, and presented the plan to a meeting of his staff. They agreed.

On about 20th January our first system (system 14!) was set up by an apprentice from Systems, and by the end of February we had all 4,000 + records loaded onto it and were producing outputs on demand — information which was needed and could not have been made available from other sources.

Not that it was that easy, of course. Not entirely to our surprise, in spite of all the time spent in "preparation", we found there was a lot to learn.

We were fortunate that fairly recently a salary planning exercise had been carried out for corporate staff on standard forms, and we were able to load from these. They carried a lot of data: name, initials, sex, date of birth, employment date, date started job, job title, grade, salary, and date, amount and reason for last increase; we also coded onto the forms location, organisation, payroll and function (activity). (It is interesting to note that now we produce these forms from the system — actually, we produce the data on labels — for the annual merit planning exercises.) We loaded the data at the terminal ourselves and using anyone we could find, onto what started off as a straight ABRA system (the precursor to MABRA).

The ABRA we started with was nothing like MABRA or PRIS is today. ABRA was essentially a simple record handling system, or so it seemed to the user. It was concise and logical; if the user got lost, he typed *HELP* for advice or *STOP* to go back a step. Most of all, it was pretty foolproof, and has since been developed to the stage where it is now almost perfect. Input was easy and (more important) output was very flexibly and readily available in two basic ways — lists and analyses. The idea was that we would put all our needs and problems to Hugh; he would sort them out for our system (then called OMP), and decide which features he would build into his own system for I.P. Sharp Associates' public library. One of the most enjoyable features of our relationship was Hugh's concern to cut costs down, and his delight whenever he found a new way to do so.

At first, every input was updating the file on line; Hugh soon fixed that. We had no validation, no coded fields; in a few days (hours?) we had them. While the validation was being brought to perfection, we were having particular problems with line noise, so a change was put in to allow input to be printed for checking before the update was filed for processing later. Also, since we had rushed into the data loading and updating and used people with absolutely no training, we found we had to check some of the work, and ways were developed to list and withdraw pending updates. We found that some of the input sheets had extra data items, so we added the fields to the system and loaded the data. Some of our coding did not fit adequately, so we changed it as we went. We found that we had specified some field sizes wrongly, so we changed them (grade went up from 2 to 3 characters to accommodate personal grades; job title and name went down from 35 to 30, and 18 to 15 respectively, because an analysis of the file showed we did not need the space).

Not content with the amount which we had loaded originally, we designed punching documents for collecting more data from a variety of forms in the seven or eight offices responsible for the corporate staff administration. We had not previously in any way taken the system to the users, and it gave us our first real taste of writing user instructions which were clear and concise and of initiating people in the mysterious art of coding data. We picked up 18 items of job history (6 items, 3 times), a dozen on qualifications, and a few odds and ends. These were merged with the main file on "Name, Date of Birth". We quickly hit problems with matching the records, particularly since some of the original input forms had not carried the full date, merely MMYY; a quick clatter of APL keys, and the problems were fixed. Even so, we still had our first error listings to deal with, and learnt some obvious lessons like:

- starting off with a small batch and a corresponding small section of the main file;
- mark each record merged, and don't allow re-merges;
- print your error list in the same sequence as your list of non-matched records;
- validate all input fully; identify invalid data on a boolean field matching the input, and accept the rest onto the file;
- whenever possible, enter absolute record numbers onto the input document, and merge on that.

Not that we learnt all these at once, of course. The delight was that we could always make enough progress fast enough not to get bogged down in systems problems — we had enough of our own.

In a similar vein, we collected and loaded addresses and postcodes. The U.K. postcode comes in a rather unusual format; we did not give clear enough instructions to our users (who could, with that?), and finished up with Hugh having to write a little routine for re-formatting, which we have used occasionally since. We also hit our first really serious problem, though we did not realise it then, with a merge not working properly. We thought we had checked the data after the merge (we probably did with the first address merge, but possibly not with the second); unfortunately, at the time we were not keeping a system diary, though I suspect that by the time we found the problem it would have been beyond redemption anyway.

These exercises were initiated through meetings with representatives of the administration offices — we held a small number of meetings, which developed understandings and guidelines on security and access, updating disciplines, transfer procedures, etc. These meetings led us to the concept of "Company Representatives", with certain clearly defined responsibilities for the corporate system. They would:

- organise the updating and maintenance of the data;
- specify the access to be allowed within their company;
- authorise access to "outsiders";
- promote the system within their company;
- advise on any proposed system developments, coding changes, etc.;
- represent the company's interest to the central team.

This form of representation has proved ideal, and has continued unchanged through a number of organisation and system developments.

Soon after we had started the merges, we also had to move into the updating business. We did this centrally at first — a good thing, since it showed us many operational problems and allowed us to devise solutions without losing our users' confidence. Hugh had been working on a command which we called Update Record (UR), which worked on the two segmented areas of the file, namely job and salary history. This simply took groups of fields and moved them down together one position in turn; soon we had additional job and salary history systems, which took the data which had dropped out of the bottom segments on the main system. As we used this command, so we modified it to cheapen it (printing the same data when each record is found, to allow the state of the record to be checked), to make it more flexible (by allowing other data to be changed at the same time), to fit our revised ideas about the segment data content, and to allow further levels to be held. Since UR works sequentially (as the data is put in, not in date order), we have had problems from time to time with backdating beyond the effective date of the latest segment already set up; had this arisen frequently, we would have devised an APL solution, but as it was, we overcame the difficulty by providing clear guidance through one and only one of the alternative update strategies. Of course, our updating of the file meant that we had to modify the routines for merging the historic data, which was still ongoing.

We had also to develop codes to reflect the new organisation, and this meant changing the size of the (various) organisation fields. The coding of changing structures of this sort is difficult, particularly when one is also trying to store the historic data. We finally decided to rely on the I.P. Sharp Associates back-up facilities if we ever wanted really accurate historic files to work from. Our code file at any time would carry a fully-structured version of the current organisation, and relevant (unique, non-continuous) parts of old organisations; the descriptions would be structured similarly. Whenever there was an organisation change, data would be changed in all the organisation fields except where we were keeping the old code codes on file, and for these we would use UR. What we still have not built in, though will do soon, is a sequential first-character coding to represent each major change of the organisation structure.

With UR, we realised that a number of our field names were useless. DOLASTINC may have been sensible as a name for the field holding the Data of the Last Increase, but with the introduction of segments, we naturally decided to number the fields in each segment 1, 2, 3, etc., and typing *DOLASTINC*1 as the first unique identification for the field became a real bore. For the first time really, we sat and thought about formats:

- field names and code descriptions should be as short as possible, but clear;
- there is no point in shortening one element any more than is necessary, as determined by the other elements and the standard output routines.

We increased the size of a number of descriptions, either because this "rule" allowed us to, or because we found them unintelligible to others (our internal postmen could not understand our 6-character locations, so we increased to 20). Some system developments encouraged us to make changes. For example, the standard output format for dates was 8 characters wide until the "age" facility arrived, when it was cut to 2; we cut *DOBIRTH* to *DOB*, saving 4 characters every time it was printed. In our efforts to be able to get more data onto a line, we even cut the spaces between one field and another in standard output formatting from 2 to 1.

When we set up UR, we had to decide how many segments to keep, and to produce guidelines on just when a change was significant enough to warrant taking up a new segment. Design and operation of historic segments is an area for big compromises. We changed our views several times as the arguments went back and forth, but we have changed the system and associated instructions very little in this respect, and then only after full consultation. As more users became involved with updating the system, so the investment cost of any change in updating increased — not a systems cost, but a human one.

As the new organisational responsibilities settled down, so we passed the data maintenance and output responsibilities out to the admin. departments in the companies. Fortunately, Hugh had by then developed MABRA — the multi-access version — so we moved swiftly into the world of access controls. The prime access was by organisation, but this was also in the job history segment, so we set up a new organisation field purely as an access key, copying the data from the main organisation field into it.

We had decided to standardise on the AJ630 terminal — easy to use, quiet, wide carriage and just about portable — but as we ploughed through our first user training session, I began to wonder whether even this was too much. Our training expert (since departed) had been mesmerised by the problems of signing on, and he spent three hours trying to teach six of us to do just that. We had even developed a special little system (TRAINING) to obviate security problems; it had all the features of the main system, but we did not need it that day. We now run our own training courses. In the first, one instructor with two terminals takes six people through some elements of the system design, security, coding, signing on, and the basic system commands; it lasts about 5 hours and gives them the confidence and knowledge to operate the system on their own. The second course is run in much the same way after two weeks' or so operation, and covers the more advanced aspects of system operation and cost reduction.

At about this time we produced the first version of our system manual. The first formal issue came six months later, in December. The manual now runs to 150 pages, of which 9 date back to 14 December 1978; the coding section is an additional 80 pages; including 20 "originals". The manual is still not complete, and it still needs improving. It is designed in sections, for easy reference, and is loose leaf; we hope that most of our users throw most of it away, since many sections will be irrelevant to them individually. It is, naturally, on a word processor. The codes, we try to take straight from the system; it gets confusing if one tries to maintain two coding tables.

In our continuing efforts to keep cost down, Hugh suggested that we should introduce a "fast search" to find individual employees. Before then we had used inverted fields whenever possible. We implemented a search facility based on NAME, INITIAL, which was very satisfactory. Some time later, Hugh developed the idea of Absolute Record Numbers, and we were able to drop the earlier approach, with further cost savings and even easier operation.

Meanwhile, work continued on the output facilities. List Record (LR) was modified to allow sorting in ascending or descending sequence. We introduced a command (Batch Print - BP) to initiate large lists at the highspeed printer; the print specification options defaulted to our needs, unless overwritten, and, to prevent formatting problems, we built in a cut-off at 132 characters. We put a similar restriction into LR, at 140 characters (the terminal output width), but allowed this to be overwritten if required; we also introduced automatic tab-setting to speed output. To our surprise, we found a heavy call on home and company address labels, and a Batch Label (BL) command was written to allow these to be initiated on demand with a number of options. A Standard Report (SR) command appeared, to allow preformatted multi-line reports to be produced on demand at either the terminal or the HSP; before then we had often got round the problem by producing a number of one-line-per-record prints, sorted in the same sequence, which we could lay out together. The Analysis (AN) command was changed quite substantially to cut costs, improve format, and give an "averaging" option.

Throughout, we were looking for consistency between commands, simplicity, reliability and cost reduction. We put in automatic updating; automatic reporting of update problems (if any) and progress on HSP requests; traps to prevent problems arising from simultaneous running of too many N-tasks or from WS FULL's; significant changes to the Add Record command; a simple means of excluding leavers from all system work unless they were specifically required; and automatic dropping of the completed input file each weekend.

Within the Company operations, a new grading structure was introduced. It was actually two structures (1-4, and A-D), and we had to code the latter +A to +D to allow the sort sequencing to work meaningfully. A field was set up to hold planned new grades; from this, salary adjustments were calculated, and held on the system for costing and subsequent automatic update to the salary segments. The new grade field was then dropped, but the salary adjustment field was used then for merit planning exercises. This allowed planned merit payments to be entered in advance, analyses and adjustments effected, payroll update lists produced, and the input to the salary segment made automatically (much easier and cheaper than our UR command). This latter advance had been made possible by the development of another new command (CD - Change records to Different values) which used some of the ideas from CR and UR.

We had always been concerned about the validity of some of the data, particularly personal details (address, qualifications, etc.), and had realised that the only way to check this was through our employees. Fortunately, we had started work on this, and so were able to pilot it along with a resourcing study being carried out within one function. As with all the output features, this is now initiated on demand from the terminal, easily and with adequate flexibility to meet local needs and peculiarities.

Growing Up - The Plant Systems

At about the time that we were working on the job history merges on the corporate system, we also started on far and away the biggest part of the task — the development and implementation of the plant systems.

Inevitably, the first plant that we set up was still not our pilot site. We were asked to move in quickly to help with a major redundancy exercise at our Liverpool plant. We decided to take what data we could from their existing computerised personnel system and worry about converting it to our standard approach later - the SPEKE system was born. To this, we merged corrections, additions, and average earnings data for redundancy calculations. Analyses were used for costing alternative strategies. In one of the more serious panics, between 7 pm and midnight one evening, I specified and Hugh wrote routines for the highly complex U.K. government and Company redundancy terms. In another panic, we worked overnight to produce individual statements of terms for all the redundant employees; being sceptical of the performance of computer systems, our users were most surprised when we delivered the goods. We even persuaded the officials of Her Majesty's Government to accept computer lists of redundant employees in place of their own forms. An important element in the success was the ability and enthusiasm of our key contact - a clerk - on the site.

Meanwhile, work proceeded on the pilot site. They also already had a computerised personnel system, though output problems had caused it to fall somewhat into disuse. Inadvisedly, we decided to update this as far as possible before loading PRIS; we weren't close enough to the ground, and the plant had put a lot of work in before we discovered that they were working from a print which was in a totally different sequence from how their manual records were stored. Once we got the system up, things moved much faster, and it has now been used for the planning and administration of two exercises of the kind which originally led to the plant being chosen as the pilot site. It is a fine justification of commitment (and horses for courses) that the personnel manager in charge of the first exercise said:

"The service we get from PRIS is as good as that from payroll is bad.".

There followed a brief period of consolidation. We put in a fast search (on CLOCK number this time) for the plant systems. We combined all the suites of programmes (OMP, TRAINING, SPEKE, PRIS) into one (PRIS) to concentrate development and problem solving. Originally, all "global" codes were maintained on the corporate system; we set up a new PRIS system for global codes, allowing full descriptions to be held in addition to the normal brief descriptions. Recently, we have installed a command for transferring records from one system to another to support interplant movements.

In the same way as we had "representatives" on the corporate side, so we introduced "creators" for the plants. We had always appreciated that there should be someone on site co-ordinating the introduction of the system. As we progressed, we realised that this was an ongoing responsibility, including:

- setting access to his system;
- ensuring security procedures are set up and followed;
- maintaining local code tables (generally only 2, compared with 22 global codes);
- ensuring that responsibilities for maintaining data are allocated and carried efficiently;
- advising, and solving problems on behalf of, his people;

- ensuring that his people are fully supported with procedures, training, etc., including appropriate Add Record options;
- maintaining equipment, paper supply, etc.;
- keeping his systems within costs;
- inverting fields as necessary;
- representing his plant to his company and the centre;
- identifying needs for local fields or other customerisation;
- promoting the use of PRIS for problem solving within his plant.

We wrote these in detail into the manual, first having cleared them with the relevant personnel managers, and developed a specific training course to back them up.

We also noted that our creators were not computer specialists, and that we were installing a standard system, so in some ways we had to protect them from the flexibility of MABRA. In particular, we prevented them from creating, erasing, re-sizing, re-blocking or copying systems; from adding or deleting fields or (other than inversion) changing field status; and from allowing others to run the update task on their systems.

As we moved into the plants we were able to consult from a fairly strong base. Not only had we senior management backing, but there was within the company an employee participation system. We discussed PRIS in participation at Cars level, then down through the divisions, before we went into the first plant; there was concern about security, about why we were using an external bureau, and that employees should be able to see the data held. In each plant, we tended to speak intially with the personnel manager, then with his management colleagues, followed by demonstrations to the personnel department and discussions with the local participation committee.

There was a difficult decision to make about who would pay for the systems; we had to weigh the central and company benefits and the fact that each plant would carry the total cost of updating and storing the data, against the need for local responsibility for holding costs down and the problems of plant needs for additional fields. Somewhat against the organisational philosophy, it was decided that we would carry the costs at Cars level (at least until the end of 1980), but that each plant would approve the specific proposals relating to it.

Throughout, we attempted to keep plants fully informed of progress. We produced a "balanced work load" plant implementation schedule, but soon found that we would make faster progress if we supported first those who, for whatever reason, were more enthusiastic. In an attempt to move faster with our limited resources, we produced a "do it yourself" guide; this proved a useful opener, but was never used by anyone on their own.

For most plants, we adopted the strategy of loading the data as far as possible from other computer systems, particularly payroll. From this data, in APL, we converted to standard formats and codes and created as much additional data as we could; even though this may have resulted in errors on many records, it reduced the updating workload later. We have automated much of this, partly by setting up a "template" - system, and Raj Chauhan (who has been manfully supporting us and has dealt with most of this rather tedious work between the more interesting aspects of the job) can normally set up a system, bells, whistles and all, within half a day; the associated programming may take another day. We would then produce, in the same sequence as the manual records and in an appropriate format, a print of all the data for checking, amending and extending. The plant would work through this sequentially, updating the records as they went; as real changes (e.g., job changes) came in, they would deal with the record totally, even though out of sequence. When this was all completed, they would then initiate the validation exercise with the employees.

In the minority of cases where existing systems did not hold sufficient data or where the population was too small to make the writing of extract programs, etc., worthwhile, we prepared documents for direct loading or for punching.

We still need to do a lot more work to tie the systems in with administration. Our biggest plant (over twice the size of the next largest) is leading the way in this respect; for example, they have a routine for producing completed payroll input punching documents for new starters.

At the time of writing (July 1980), we have well over 80,000 employees on the systems. Most of the remaining plants are small, isolated and without landlord/tenant complexities (i.e., where employees are based on the site but do not report to the plant director). For these, we are working on extending the payroll systems to hold more personnel data and then setting up the data on PRIS without subsequent amendment, on a regular basis such as quarterly or half-yearly; this will provide all the essential output features and most of the benefits for a much lower cost and plant workload.

Maturity - Consolidation, Further Developments, and Effort Reduction

We have come a long way, as the preceding (but by no means complete) description shows, but there is still a lot to do. The efforts will continue to be directed to improve facilities, improve service, cut workload and cut costs.

As it is, we have produced a redundancy calculation and administration program; this works off the file (but can stand alone) and meets the total range of needs required by both company policy and specific local variations. A wastage analysis package (WASP - originally developed by the Institute of Manpower Studies) is nearing completion; one for career structures (PROSPECT, also from the IMS) will hopefully follow. These are essential manpower forecasting tools, which the present economic uncertainty makes even more desirable.

We have developed a routine for selecting and printing records from different systems together in one hit, and are working on the same for analyses - particularly useful for company and Cars staffs.

One aspect that still could do with improvement is the turn-round time for HSP's. We seem to run into some ridiculous problems with delivery from I.P. Sharp's office in London to our plants. Possible solutions are the planned installation of a printer in I.P. Sharp Associates' Coventry Office or perhaps the development of some form of direct link from London to BL Systems' data centre.

Many standard reports and analyses are waiting to be specified and written. Similarly, we need to put in the automatic, regular production of some reports. We took a conscious decision to concentrate on implementations first. I am sure that if all PRIS output had to be produced centrally, we would have put effort into this much sooner, and in some ways this would undoubtedly have been beneficial; however, the system is quite flexible enough to take this need away. Since the system is so complex and report formatting relatively trivial, this is one of the few areas which we should be able to cover by developing APL expertise in our own department.

One of the areas we have been talking about for some time is the provision of some sort of improved transaction reporting/analysis facility. This would pick up all the changes actioned within a time period, probably by using a boolean field. It is just a matter of finding time to specify it.

There is a continuing need to improve our communication, education and support services. We keep working at the manual; we are looking at running modules of the advanced/further advanced course; we would like to hold regular (e.g., 6-monthly) user conferences; we need to do more on our creator/expert course. We also intend to establish regular audits, to help the plants. And we need to do a lot more to educate potential recipients of information about what they really could learn about manpower in the business, if they would just ask.

Finally, 1981 should see the completion of payroll standardisation, and we are considering, at least for the hourly paid, moving to the payroll-based approach described earlier which, it will be recalled, we would have done from the start, had this been possible.

The Control Systems - The PRIS Team

It should be obvious from what has gone before that the role of the centre is not just to install the system, nor even just to develop it. The centre is a service, and in particular, it must:

- generate confidence;
- provide training and guidance;
- give prompt resolution to any problems or needs;
- identify new needs and opportunities, and devise and communicate appropriate strategies;
- support the users in promotion of the information system.

It also has certain other responsibilities, in respect of system integrity and cost control. It must be an arbiter and a decision maker. It must keep close enough to major organisation developments and needs to identify (and, if necessary, co-ordinate) potential uses for the system.

The PRIS team set out to meet these responsibilities and provide the service. Through 1978 and the first half of 1979, we probably averaged 7 man-days/week working on PRIS, and from then to mid 1980, including the training of new team members, we averaged about 16; secretarial support, and any occasional help we could scrounge from anywhere, was additional. I.P. Sharp expert programming support averaged about 3-4 man-days per week throughout this period, and BL Systems Limited provided about the same from the beginning of 1979 in data gathering.

We organised ourselves to meet this by allocating particular people to look after specific plants, companies and codes, while keeping system development largely in the hands of just one. We are now looking at broadening our operational contact base by also giving these people responsibility for promulgating manpower planning concepts and by getting them involved in the business planning process, thus building on the information system.

The Obituary

At the end of the day, and I think that is a long way away, how will people judge it? I believe that they will say that those concerned, with a lot of luck and some hard work, with a bit of risk taking and decisions on the fly, by creating opportunities and having a responsible attitude, developed a system (MABRA) and an aid (PRIS) of benefit and credit to the firms involved. And that should be enough.

MOTTO OF A FRENCH LYCEE:

> "Common sense with logic : superior
> Common sense without logic : inferior
> Logic without common sense : disaster"

ROBERT FROST:

> "I have promises to keep and miles to go before I sleep"

MOTTO OF THE OUTWARD BOUND TRUST:

> "To serve, to strive and not to yield"

P.B. SHELLEY (Out of context, one hopes!):

> "Look on my works, ye mighty, and despair!"

JOHN KEATS:

> "A thing of beauty is a joy for ever:
> Its loveliness increases; it will never
> Pass into nothingness ....."

And the all singing and dancing system? The pilot started up in June this year — but that is another story.

Acknowledgements

BL CARS:
The PRIS team: (who made it happen)
Frances Wright, Sandra Fletcher, Tony Tattum, Ian Williams, Marion Hayes
The Management, for their support
The users, all of them, for their hard work, ideas and patience.

BL SYSTEMS LIMITED:
Trevor Towe and his colleagues, for getting the data.

I.P. SHARP ASSOCIATES LIMITED

Hugh Hyndman, without who it would not have been
Raj Chauhan, without whom it would not have continued
The operations people, for performing so well against all the odds.

INSTITUTE OF MANPOWER STUDIES:

Malcolm Bennison and his colleagues, for their encouragement.

ALL OTHERS, who know who they are, for whatever reason.

This paper represents the personal views of the author. It does not necessarily reflect the views of BL Limited, I.P. Sharp Associates Limited, or any other organisation or person mentioned therein.

COALMOD: A FINANCIAL ANALYSIS AND POLICY SIMULATION MODEL FOR COAL MINING DEVELOPMENTS

Frank C. Basham
Philippe Monier
B.C. Ministry of Energy, Mines and Petroleum Resources
Victoria, British Columbia

Abstract

The paper outlines the scope and applications of "COALMOD", the Ministry of Energy, Mines and Petroleum Resources' computerized financial simulation model for coal mining developments. After a brief description of the computer system and language of implementation, the logic and scope of current model sub-routines are presented. This is followed by a presentation of examples on the current applications of the model to public policy evaluation including benefit-cost analysis, fiscal analysis, and evaluation of the effects of public infrastructure investments made in support of mining projects. The model is presently in a "development" stage and refinements are continually being introduced. Some examples of proposed refinements are given.

Introduction

This paper has been developed in response to a number of requests from coal mining companies and other parties who have indicated interest in understanding our particular application of APL computer modelling to coal industry financial analysis and simulation. In particular, the subject of this paper is COALMOD, a mnemonic given to a computerized financial simulation model for coal mines. This model is assisting the Government of British Columbia in evaluations of new coal projects in the province; in assessment of public sector investments made in support of these projects; more generally, in the measurement of current or proposed government policies; and in measurement of the effects of changing factors such as coal prices and transportation costs, largely outside the immediate government control.

The relevance of such computer assisted financial simulation to the coal industry and more generally to the energy business in this country should be readily apparent to this audience. Energy is in the news every day. While parts of North America are abundantly blessed with supplies of conventional crude oil, natural gas, hydro electricity and coal, other regions are chronically deficient and/or chronically dependent on high-priced imported energy. As prices of the conventional fossil fuels continue to rise, substitute fuels, including coal and products from coal liquefaction, become more economically attractive to consumers, the coal industries on Canada's east coast and in western Canada are beginning to capitalize on this emerging domestic energy market. As well, over the past ten years, higher quality metallurgical coals for use by the steel industry have made a significant penetration into Pacific Rim, European and South American markets.

As domestic and international coal markets continue to expand, there has been substantially increasing pressure on exploration and producing companies to expand their production capacity. This attendant pressure is also felt by government agencies who are in the business of allocating rights to the coal resource, in deciding the appropriate tax environment for development of the industry, and in approving the multitude of environmental requirements associated with this type of mining development. While British Columbia's history of coal development dates back into the last century, it is only in the past two decades that coal has regained the prominence it once had. At present there are only five producing coal mines in B.C. but the potential developments now number nearly two dozen active proposals. With this magnitude of impending development facing British Columbia and other western provinces, a rigorous and systematic evaluation procedure appeared to be necessary to various government agencies.

COALMOD was initiated in 1976 in response to requirements for a contemporary computer based discounted cash flow rate of return (DCFROR) analytical tool to assist in the evaluations of proposed coal projects in Northeastern British Columbia. Initial development of the model was financed jointly between the B.C. Government and the Canadian Government. It was first programmed and implemented in APL for use on the B.C. Systems' Corporation Honeywell computer. In 1978, the model was revised and changed over to the I.P. Sharp APL system due to deficiencies in APL related computer support by the system's previous host architecture.

Since the first COALMOD runs in late 1976 and early 1977, the model has been substantially improved and its range of application expanded to include not only mine-specific financial analysis, which was its original application, but also benefit-cost analysis of coal projects and other applications described in the last part of this presentation.

We will now proceed with the bulk of our presentation which is concerned with the general system overview including our rationale for selection of APL, the essence of COALMOD, a representation of the system and its modules, a discussion of special calculation features, illustration of some of the reports available, some examples of proposed new model developments or enhancements, and finally, a discussion of the current areas of application to public policy.

COALMOD Computer System Aspects

Why APL?

COALMOD is programmed in APL and stored on-line with I.P. Sharp Associates' (IPSA) APL system. APL software support, data management and security are also IPSA products. As a computer language for COALMOD, APL was selected primarily because of the range and power of its primitive operators and consequently the ease with which non-programming professionals can use it interactively to obtain immediate results. Also, the manipulative capability of this language for data in matrix or array form is superior to most other languages.

Essence of COALMOD

The coal cost model is an economic and financial flows model. It is used as an analysis tool when examining different sensitivities between several options of a coal mining operation.

In simulation of fiscal policy effects, the model's available tax routines can be used to evaluate different tax and royalty rates and/or structures (e.g., an Alberta coal taxation environment).

The model is used to calculate net discounted cash flows, rate of return on investment over the life of the project, and payback period. Coal price and coal production for both thermal and metallurgical coal determine the revenue part of the cash flow in while owner's equity and borrowing determine the financing part of the cash flow in. Cash flow out constitutes capital investments, production costs, taxes and royalties.

An important purpose of this simulation model is to estimate tax revenues to the municipal, provincial and federal sectors.

For cases where some of the necessary data (particularly costs and capital investments) are either not available or unreliable, the model has been provided with a capacity to generate the missing parameters internally using formulae based on statistics obtained from operating coal mines.

At the present time, the scope of the model is restricted to the production of deterministic financial results. By changing many of the input vectors or scalars, it is currently possible to provide some sensitivity analysis capability.

Base case runs can initially be made using, for example, constant prices, current tax rules, one hundred percent equity financing and no inflation or contingencies. These base assumptions can be gradually relaxed to reflect more realistic circumstances; variations from the base case are then measured. Of course, a base case can be defined differently.

A combined total of more than eighty scalars and vectors are "input" for the model (see Exhibit 3). These include the mine operational variables such as coal production by year, revenue and operating cost by year, the capital cost parameters, and a variety of scalars used to select the desired tax and royalty systems.

System Overview

The COALMOD computer system is maintained and accessed through the Sharp file system. The computer system contains the calculation functions, print reports and general user support routines. A System Overview flow chart is shown in Figure 1.

To satisfy security requirements (e.g., of various input data sets) a stand-alone computer filing system has been designed to operate within the COALMOD computer system. This CASE system, as it has become known, allows the user to add CASES (data sets of inputs and outputs), replace CASES, delete CASES and to selectively allow other users (other computer account numbers) to read or read and write selected CASES. This CASE system is similar to the I.P. Sharp STARS system. Of course, there may be several CASES for any one coal project.

For each version of the model (production and development) there are two separate workspaces. One is considered to be the users' workspace and the other is the programmer analysts' workspace. In addition, for each version there is a systems file which stores all the functions and variables required to support the model.

Although there are a large number of user interface routines, system filing support routines, a CASE security system, and various input and output routines, the true core of COALMOD is the set of calculation functions that takes the input data and calculates the output data.

A short illustration of how the user can read a case, perform all calculations and send a report to a remote location is shown in Exhibit 1.

The Modules

To perform all the calculations in COALMOD, the user has only to enter CALC. This function invokes several other functions which calculate all cash flow items. These functions have been grouped in nine different logical modules. The following explains briefly each of these modules, in order of their sequence and execution in CALC. For each module the type of action is indicated followed in parentheses by the name of the general function to which it corresponds.

Module 1 - **Driver (CALC)**: prepares conditions for executing the Model module and calls it.

Module 2 - **Model (FNECSF)**: coordinates the model calculations but does not actually perform them.

Module 3 - **First Vector Calculation (FQSETUP)**: performs all the calculations which can proceed in parallel prior to the tax calculation.

Module 4 - **Operating Cost (FCO)**: calls several modules which calculate the various costs of operation and computes several sums of these quantities.

Module 5 - **Scalar Initialization (FINITY)**: since the calculations done inside the loop are done with scalars, the scalars must be initialized every year to the value of the corresponding vector in that year.

Module 6 - **Administration and General Cost (FCA)**: first calculates property taxes and provincial capital tax, and then from these calculates administration and general cost in the default equation if they are not provided.

Module 7 - **Tax Calculation (FT)**: performs all direct tax calculations in accordance with the tax option selected. Calculates first order taxes: federal corporation income tax, B.C. corporation income tax, B.C. mining tax on income and B.C. coal royalty/mineral land tax.

Module 8 - **Scalar Value Saving (FSAVE)**: since the modules inside the loop of the Model module use the scalar variables year after year, the values of each of these variables must be saved as a vector for printout at the end of a run.

Module 9 - **Second Vector Calculation (FENDCALC)**: performs remaining calculations using vector processing. This automatically runs at the end of the loop in the Model module when all calculations done with scalars are complete. In this module, second and third order taxes are calculated from a set of built-in algorithms and include the following:

- taxes (other than on income) applying to corporations are second order taxes like federal and provincial sales taxes, B.C. Corporation capital tax and municipal and property taxes, and
- federal and provincial taxes on employee personal income are third order taxes.

Special Calculation Features

One major logic feature in the model deserves special mention. There are three cost computation branches or options in COALMOD (see Figure 2).

The first branch, which may be described as the default case, is used when there is limited cost information or when a project is in a very preliminary stage. In this branch, preliminary geological and engineering data are entered (annual production, stripping ratio, clean coal yield, fines yield, seam thickness, overburden depth, etc.). Capital and operating costs are derived endogenously from default equations in the model. These equations were derived from generalized engineering cost equations modified for existing B.C. experience. In practical terms, this branch of the model produces the least reliable results.

A second branch or data option for the model uses company supplied or feasibility study estimates of various capital cost items and total estimated operating costs (but no details). The default equations described above are overridden in this option. This option is the simplest to use and provides near perfect replication of a company's own results. Thus, COALMOD in this application, is primarily a system which manipulates data into an output report and enables "sensitivity" calculations to be made.

The third branch is a variant of the second involving the inputting of detailed operating costs or costs per tonne associated with each cost centre (open-pit mining costs, waste removal costs, trucking costs, washing and loadout costs, administration and head office expense, etc.). This branch enables better sensitivity calculations to be made.

In addition to these three main cost computation branches in COALMOD, there are a variety of sub-options which deserve passing mention:

1. routines for one pit or two pits;
2. routines for underground and/or surface mining;
3. routines for metallurgical or metallurgical and thermal coal production.

Also, a fairly standard set of financial or tax variable options can be set:

4. debt to equity ratio;
5. interest rate on capital borrowed;
6. discount factor;
7. all tax rates are variable, although deduction/CCA rates and the like are constant;
8. inflation rates can be applied independently to capital and operating costs and prices;

9. capital cost contingency factor.

Special ROR Function

In some instances a user may wish to calculate which minehead price will result in a predetermined (say 12%) after tax rate of return on owners' equity.

ROR does this when the user has provided two starting prices (normally upper and lower bounds for the price search) and the maximum number of iterations.

A summary printout will automatically be printed for each price used by the routine on its path to convergence. Lastly, a basic plot of price versus rate of return will be produced.

The Reports

Once the driver function CALC has been called all calculations are performed. Then the function REPORT will allow the printing of the desired report(s).

The reports available are:

1. A summary of key results over the life of the project: see Exhibit 2. This summary report shows the pre-tax rate of return on investment, the after-tax rate of return on owners' equity, the payback period and the total life-of-project cash flows, average annual flows, or per tonne flows for the salient revenue, financing, operating cost, capital investment and tax items.

2. A list of all inputs as entered, i.e., their current status before being altered by the model (in accordance with input changes from the user): see Exhibit 3.

3. A list of all inputs after being altered by the model, i.e., inputs adjusted by inflation factors, contingency factors, etc. as well as substituting default values/calculations when the user has not supplied a particular data input. e.g., Working Capital is calculated by default to be 15% of Total Production Costs (after their adjustment by inflation factors).

4. A very detailed report of all input and output data. These are presently grouped in several tables according to their nature. Their grouping can very easily be modified with minimal programming work.

   Presently, the detailed report shows the following tables:

   a. All Scalars
   b. Parameters
   c. Production
   d. Revenues
   e. Expenses
   f. Cash Flows
   g. Loan Schedule
   h. Factor Cost
   i. Capital Investment (for CCA Balances)
   j. Federal Taxes
   k. Provincial Taxes

New Developments

We have investigated the possibility of building additional model components for the estimation of reserves, detailed estimation/simulation of key cost centres (e.g., truckhaul or underground haulage), and equipment and labour productivity given different geological or structural conditions. We have also looked at components which would assist in the evaluation of new technologies or environmental regulations. At present, these refinements are still under investigation. Any analyses along these lines must presently be conducted outside the scope of the model and translated into operating cost estimates for input to the model. There are several excellent examples of complete coal project evaluation systems which are guiding our thinking in this regard.

Another main area of programming development is the refinement of the tax minimization process. This of course, has to be flexible enough to allow the consideration of future tax rules.

Current Applications

In the government sector, the primary uses for computer based evaluation systems like COALMOD include benefit-cost analysis of proposed projects, evaluation of public infrastructure investments, and measurement of the financial effects of public policy affecting the coal industry. The latter includes examination of the structure and burden of taxation on the coal industry in British Columbia.

Benefit-Cost Analysis

Under British Columbia's "Guidelines for Coal Development", a proposed coal project may be subject to benefit-cost evaluation if any or all of the following circumstances prevail:

- where a project generates significant environmental effects (externalities);
- where a project generates significant social or economic effects;
- where public investments for transportation, power, or townsite infrastructure are required.

In these circumstances, companies are required to submit detailed financial information on their project such as to enable evaluation of commercial and socio-economic viability. The financial information supplied constitutes part of the input requirements for COALMOD. The model produces estimates of commercial viability such as revenues, detailed cash flows, rate of return and payback period as well as estimates of public sector tax flows, a first measure of the public benefits associated with the project. Model cash flow results are then used in a benefit-cost analysis framework which incorporates private sector impacts from COALMOD, quasi-public sector impacts from road, rail, port and townsite financial analysis models and also public sector impacts including taxes, environmental, socio-economic and related estimates of impact.

Public Infrastructure Investment

The Government of British Columbia has recently established a policy of financial assistance for provision of major infrastructure for resource development projects. This policy includes rail, port, road and townsite investments necessary to the development of mining projects. Where a project demonstrates financial need as measured by COALMOD's results on discounted cash flow rate of return and demonstrates the generation of positive net benefits to B.C. and Canada, it may be eligible for such financial assistance.

"Financial Need" and "Benefits" are as measured by DCFROR and tax revenues from COALMOD. In these evaluations, the Government seeks verification that there is legitimate need, i.e., that a project could not proceed without infrastructure investment from the public sector and that net benefits to government from the project exceed requested outlays for public infrastructure. The amount of financial assistance available is tied to the number of mining jobs created by the project, to revenue constraints of the federal and provincial governments, and to cost sharing agreements under negotiation between the two governments.

Public Policy Evaluation

Our ministry's mandate is to manage the mineral resource development and production within British Columbia. This includes responsibility for the disposition of mining rights, the regulation and monitoring of exploration activity conforming to sound environmental, engineering and safety practice, for regulation and control of safety and environmental practice in producing mines, and more generally for policies which lead to optimum benefits to B.C. derived from mining activity. Systems like COALMOD for the coal industry and MINSIM, a sister model to COALMOD being used for metal mining activity, aid the government in this management function.

In addition to the public policy aspects concerning infrastructure financing and benefit-cost analysis cited above, COALMOD has been used in conjunction with other analyses to evaluate the effects of existing or proposed tax legislation. In 1978, the B.C. coal royalty system was changed from a fixed per tonne royalty of $1.48 for metallurgical coal and $.74 for thermal coal to one based on the net minehead value of coal. In this instance, COALMOD produced life-of-project financial impacts of the proposed change. As can be seen from the discussions of the major features of the system, tax rates, allowances, and many other parameters can be changed to measure the impact of different features of the tax system.

COALMOD can also be used to simulate the financial effects of other types of public policies. For example, a change in environmental regulation causing a variation in operating costs can be effectively monitored by the system. Also, as illustrated in earlier sections of this presentation, factors outside the direct control of governments such as product price and transportation costs can be evaluated with respect to financial effect on the mining operations.

Summary

In this presentation, we have attempted to illustrate how APL has been used for economic and financial analysis of coal mining projects. While such computer-based techniques do not replace sound judgement and expertise of seasoned analysts, they certainly provide efficient tools for the planning policy and decision making processes.

In closing, we wish to mention that government clients are presently the major beneficiaries of the service applications for COALMOD. However, we have made and continue to make the system, exclusive of data, available to mining companies, the industry associations and others who may not have access to this type of service. In these instances we perform runs for other agencies and companies after receiving client supplied data inputs.

```
     CASEREAD 49                                  Reads a case into the workspace.
CASE 49 SUCCESSFULLY READ
NAME: COAL MINING PROJECT



     CALC                                         Does the calculations and
CALCULATIONS ARE DONE                             selects a name for this run.
RUNNAME: METALLURGICAL COAL - OPEN PIT



     STATE                                        Displays the state.
HIGHSPEED                                         - printed on a highspeed printer.
AUTOMATIC                                         - automatic page alignment.
PAGELENGTH 66                                     - pagelength is 66 lines.



     REPORT                                       Selects a report(s) and
REPORT NUMBERS: 1                                 submits an HSPRINT.
DESTINATIONS (TO,CA,VA OR VI): TO
DELIVERY INSTRUCTIONS FOR TORONTO
MAIL TO XYZ CORPORATION,
        ABC STREET,
        TORONTO.

REQ.NO 121381 FILED 3960090 0.55.06   3 AUG 1980
```

Running a Case

Exhibit 1

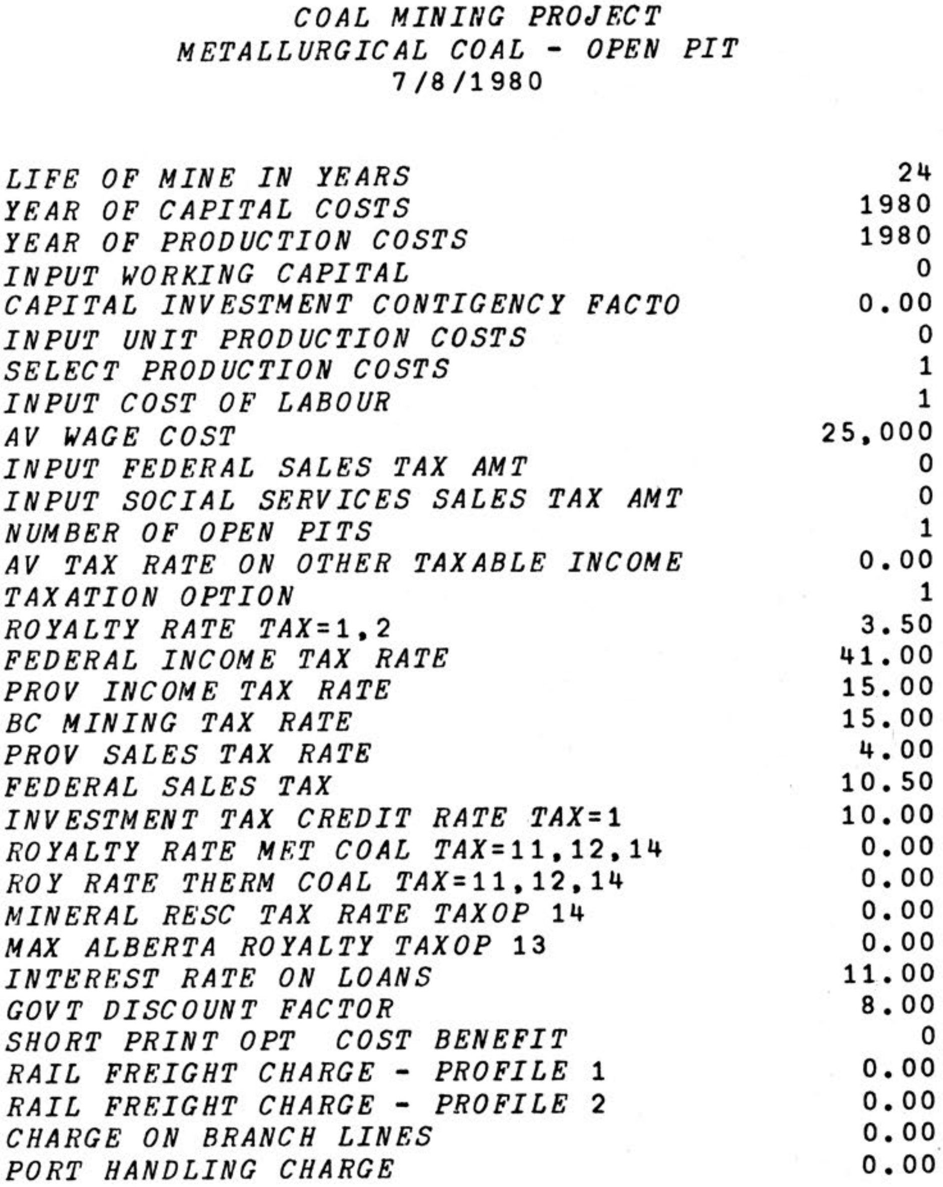
```
                                           COAL MINING PROJECT
                                      METALLURGICAL COAL - OPEN PIT
                                                7/8/1980
                                                SUMMARY

PRE-TAX RATE OF RETURN ON INVESTMENT 15.10 PERCENT
RATE OF RETURN ON OWNERS EQUITY      11.01 PERCENT
PAYBACK PERIOD                       14    YEARS
YEAR OF CAPITAL COST DOLLARS         1980
YEAR OF PRODUCTION COST DOLLARS      1980
TAX OPTION  1

                                             TOTAL       AVERAGE  YEARS   PER TONNE OF      DISCOUNTED
                                                         PER ANNUM          CLEAN COAL           TOTAL


TOTAL CLEAN COAL                        86,000,000       4,300,000    20            1.00

CASHFLOW IN

GROSS REVENUE (CASHFLOW IN)          5,599,326,056     279,966,303    20           65.11  1,940,874,657
EQUITY CAPITAL                         335,651,486      37,294,610     9            3.90    275,240,762
BORROWING                               33,995,966       8,498,991     4            0.40     23,635,482

CASHFLOW OUT

TOTAL PRODUCTION COSTS               2,693,317,372     134,665,869    20           31.32    972,037,771
TAXES AND ROYALTIES                  1,159,207,876      57,960,394    20           13.48    304,773,746
NET CASHFLOW                           844,766,336      35,198,597    24            9.82    101,180,952
CAPITAL INVESTMENT                     843,895,231      35,162,301    24            9.81    538,501,349

TAXES

B.C. MINING TAX                        231,261,118      11,563,056    20            2.69     58,293,250
ROYALTIES/MINERAL LAND TAX             195,976,412       9,798,821    20            2.28     67,930,613
PROVINCIAL INCOME TAX                  241,606,129      13,422,563    18            2.81     59,296,190
TOTAL PROV INC + RES TAXES             668,843,659      33,442,183    20            7.78    185,520,053
PROVINCIAL CAPITAL TAX                  11,313,970         595,472    19            0.13      3,743,788
PROPERTY TAXES                          42,427,389       2,233,020    19            0.49     14,039,206
SOCIAL SERS / SALES TAX                 29,175,242       1,215,635    24            0.34     11,961,495
PROV PERS INCOME TAX                    79,384,357       3,307,682    24            0.92     36,056,247
GRAND TOTAL OF PROVINCIAL TAX          831,144,618      34,631,026    24            9.66    251,320,790

FEDERAL TAX PAYABLE                    490,364,217      24,518,211    20            5.70    119,253,693
FEDERAL SALES TAX                       59,174,679       2,958,734    20            0.69     22,448,695
FEDERAL PERSONAL INCOME TAX            233,483,403       9,728,475    24            2.71    106,047,785

OPERATIONS

NUMBER OF EMPLOYEES                         24,030           1,202    20
COST OF LABOUR                         600,750,000      30,037,500    20            6.99    224,622,703

COST OF SUPPLIES AND SERVICE         2,092,567,372     104,628,369    20           24.33    747,415,068

CAPITAL INVESTMENT

INVESTMENT IN EXPLORATION               20,352,005       6,784,002     3            0.24     19,239,270
INVESTMENT IN DEVELOPMENT              279,838,984      69,959,746     4            3.25    229,679,534
INVESTMENT IN EQUIPMENT                139,604,772      27,920,954     5            1.62     87,318,220
INVESTMENT IN PROCESSING ASSETS        129,522,475      25,904,495     5            1.51     77,809,039
INVESTMENT IN STRUCTURES                43,049,424       8,609,885     5            0.50     26,458,360
REPLACEMENT CAPITAL                    210,296,643      11,683,147    18            2.45     73,037,077
TOTAL (MSA)                            822,664,302      34,277,679    24            9.57    513,541,500
INVESTMENT IN RAILSPURS                 21,230,929       7,076,976     3            0.25     14,465,282
INVESTMENT IN HOUSING STOCK                      0               0     0            0.00              0
TOTAL INVESTMENT COST                  843,895,231      35,162,301    24            9.81    528,006,782
```

Summary Report - Case Results

Exhibit 2

Entered Inputs Report

Exhibit 3 (Part 1 of 3)



```
                    COAL MINING PROJECT
              METALLURGICAL COAL - OPEN PIT
                         7/8/1980


LIFE OF MINE IN YEARS                                24
YEAR OF CAPITAL COSTS                              1980
YEAR OF PRODUCTION COSTS                           1980
INPUT WORKING CAPITAL                                 0
CAPITAL INVESTMENT CONTIGENCY FACTO                0.00
INPUT UNIT PRODUCTION COSTS                           0
SELECT PRODUCTION COSTS                               1
INPUT COST OF LABOUR                                  1
AV WAGE COST                                     25,000
INPUT FEDERAL SALES TAX AMT                           0
INPUT SOCIAL SERVICES SALES TAX AMT                   0
NUMBER OF OPEN PITS                                   1
AV TAX RATE ON OTHER TAXABLE INCOME                0.00
TAXATION OPTION                                       1
ROYALTY RATE TAX=1,2                               3.50
FEDERAL INCOME TAX RATE                           41.00
PROV INCOME TAX RATE                              15.00
BC MINING TAX RATE                                15.00
PROV SALES TAX RATE                                4.00
FEDERAL SALES TAX                                 10.50
INVESTMENT TAX CREDIT RATE TAX=1                  10.00
ROYALTY RATE MET COAL TAX=11,12,14                 0.00
ROY RATE THERM COAL TAX=11,12,14                   0.00
MINERAL RESC TAX RATE TAXOP 14                     0.00
MAX ALBERTA ROYALTY TAXOP 13                       0.00
INTEREST RATE ON LOANS                            11.00
GOVT DISCOUNT FACTOR                               8.00
SHORT PRINT OPT  COST BENEFIT                         0
RAIL FREIGHT CHARGE - PROFILE 1                    0.00
RAIL FREIGHT CHARGE - PROFILE 2                    0.00
CHARGE ON BRANCH LINES                             0.00
PORT HANDLING CHARGE                               0.00
```

Exhibit 3 (Part 2 of 3)

COAL MINING PROJECT
METALLURGICAL COAL - OPEN PIT
7/8/1980

YEAR	INVESTMENT IN EXPLORATIO	INVESTMENT IN DEVELOPMEN	INVESTMENT IN EQUIPMENT	INVESTMENT IN PROCESSING ASSETS	INVESTMENT IN STRUCTURES	REPLACEMEN CAPITAL	INVESTMENT IN RAILSPURS	INVESTMENT IN HOUSING STOCK	CAPITAL INVESTMEN INFLATION RATE	WORKING CAPITAL	OWNERS' EQUITY
----	----------	----------	----------	----------	----------	----------	----------	----------	---------	----------	----------
	1	2	3	4	5	8	7	6	40	9	10
	VCIX	VCID	VCIQ	VTCIP	VCIS	VCIR	VCIRS	VCIHS	VNIF	VCIW	VE
1980	10,000								1.00		10,000
1981	5,000	40,000							1.00		50,00:
1982	5,000	80,000							1.00		90,000
1983		100,000							1.00		120,000
1984		50,000	20,000	10,000	5,000		5,000		1.00		70,000
1985			30,000	10,000	5,000		10,000		1.00		
1986			20,000	30,000	10,000	10,000	5,000		1.00		
1987			30,000	30,000	10,000	10,000			1.00		
1988			30,000	40,000	10,000	10,000			1.00		
1989						10,000			1.00		
1990						10,000			1.00		
1991						10,000			1.00		
1992						10,000			1.00		
1993						10,000			1.00		
1994						10,000			1.00		
1995						10,000			1.00		
1996						10,000			1.00		
1997						10,000			1.00		
1998						10,000			1.00		
1999						10,000			1.00		
2000						10,000			1.00		
2001						10,000			1.00		
2002						10,000			1.00		
2003						10,000			1.00		

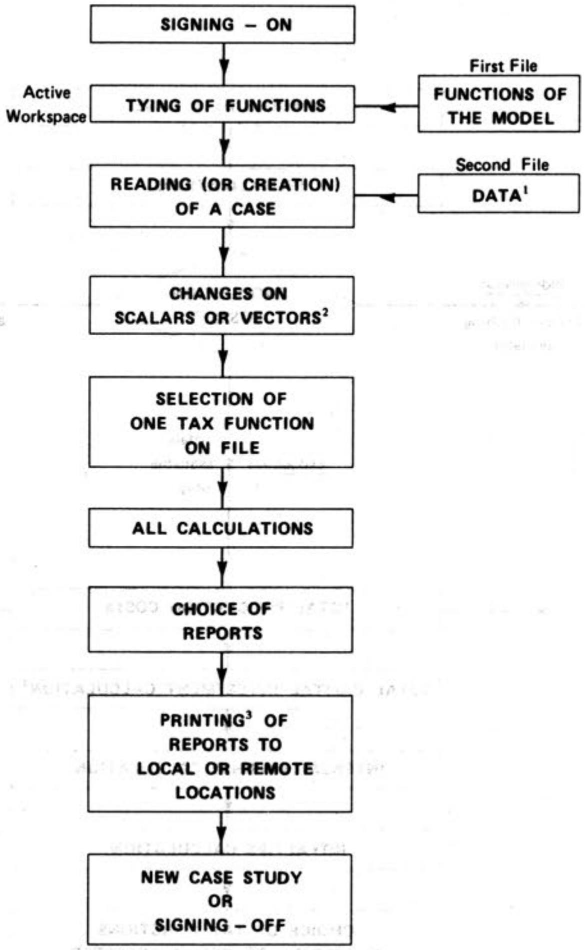
YEAR	OPEN PIT MINING COSTS	OPEN PIT WASTE REMOVAL COSTS	PREPARATI COSTS (DRYING)	UG MINING COST ROOM AND PILLAR	UG COST HYDR OR LONG WALL	TRUCKING COSTS	TOTAL PRODUCTION COSTS	OPERATING COSTS INFLATION RATE	ADMIN AND GENERAL COSTS	ADMIN COST INFLATION RATE	COST OF LABOUR	FEDERAL SALES TAX
----	---------	---------	---------	---------	---------	---------	----------	---------	---------	---------	----------	----------
	11	12	13	38	31	32	35	41	14	42	39	46
	VCOOMMT	VCOONWM	VCODT	VCUCR	VCUHR	VCTCT	VCP	VNOF	VCAT	VNAF	VCOL	VTFS
1980												
1981												
1982												
1983												
1984							64,000	1.50			13,500	
1985							80,000	1.50			20,250	
1986							96,000	1.50			31,500	
1987							120,000	1.50			31,500	
1988							120,000	1.50			31,500	
1989							120,000	1.50			31,500	
1990							120,000	1.50			31,500	
1991							120,000	1.50			31,500	
1992							120,000	1.50			31,500	
1993							120,000	1.50			31,500	
1994							120,000	1.50			31,500	
1995							120,000	1.50			31,500	
1996							120,000	1.50			31,500	
1997							120,000	1.50			31,500	
1998							120,000	1.50			31,500	
1999							120,000	1.50			31,500	
2000							120,000	1.50			31,500	
2001							120,000	1.50			31,500	
2002							120,000	1.50			31,500	
2003							120,000	1.50			31,500	

Exhibit 3 (Part 3 of 3)

COAL MINING PROJECT
METALLURGICAL COAL - OPEN PIT
7/8/1980

YEAR	SOCIAL SERS / SALES TAX	OP METALLURGI COAL PIT 1	OP THERMAL COAL PIT 1	STRIPPING RATIO PIT 1	OP AV SEAM THICKNESS PIT 1	OP YIELD CLEAN COAL PIT 1	OP METALLURGI COAL PIT 2	OP THERMAL COAL PIT 2	STRIPPING RATIO PIT 2	OP AV SEAM THICKNESS PIT 2	OP YIELD CLEAN COAL PIT 2	YIELD OF FINES IN CLEAN COAL
	47	16	17	18	19	20	21	22	23	24	25	15
	VTPS	VQOCCM	VQOCCT	VQORS	VQONS	VQOYCC	VQOCCM2	VQOCCT2	VQORS2	VQONS2	VQOYCC2	VQYF
1980					5.00							
1981					5.00							
1982					5.00							
1983					5.00							
1984		2,000		5.20	5.00	60.00						45.00
1985		3,000		5.20	5.00	60.00						45.00
1986		4,500		3.40	5.00	60.00						45.00
1987		4,500		3.40	5.00	60.00						45.00
1988		4,500		2.80	5.00	60.00						45.00
1989		4,500		2.80	5.00	60.00						45.00
1990		4,500		2.80	4.80	65.00						45.00
1991		4,500		2.80	4.80	65.00						45.00
1992		4,500		2.80	4.80	65.00						45.00
1993		4,500		2.80	4.80	65.00						45.00
1994		4,500		2.80	4.80	65.00						45.00
1995		4,500		2.80	4.80	65.00						45.00
1996		4,500		2.80	4.80	65.00						45.00
1997		4,500		2.80	4.80	65.00						45.00
1998		4,500		2.80	4.80	65.00						45.00
1999		4,500		2.80	4.80	65.00						45.00
2000		4,500		2.80	4.80	65.00						45.00
2001		4,500		2.80	4.80	65.00						45.00
2002		4,500		2.80	4.80	65.00						45.00
2003		4,500		2.80	4.80	65.00						45.00

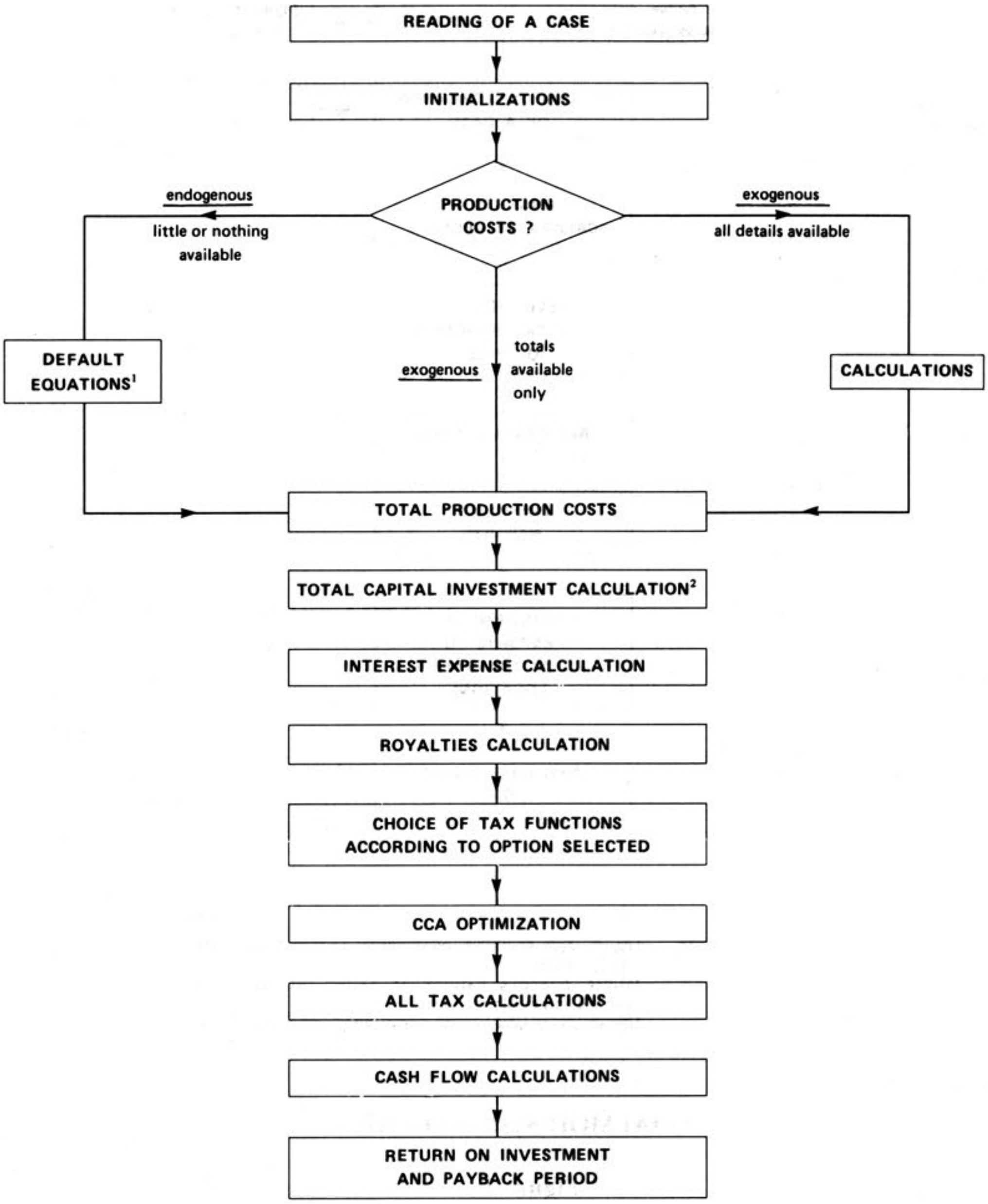
YEAR	UG CLEAN COAL PROD CONT	UG YIELD CLEAN COAL CONT	TOTAL UG RAW COAL	UG CLEAN COAL PROD HYDR	UG YIELD CLEAN COAL HYDR	PRICE OF METALLURG COAL	METALLIC COAL PRICE INFLATION RATE	PRICE OF THERMAL COAL	THERMAL COAL PRICE INFLATION RATE	HEAD OFFICE EXPENSES	OTHER TAXABLE INCOME	ALBERTA ROYALTY INFL RATE TAXOP 13
	26	27	28	29	30	33	43	34	44	36	37	45
	VQUCCC	VQUYCC	VQUCR	VQUCCH	VQUYCH	VPPTCCM	VNPCMF	VPPTCCT	VNPCTF	VCH	VTOTI	VTPIF
1980										2,000		
1981										2,000		
1982										2,000		
1983										2,000		
1984						44.00				2,000		
1985						44.00	3.00			2,000		
1986						44.00	4.00			2,000		
1987						44.00	6.00			2,000		
1988						44.00	7.00			2,000		
1989						44.00	5.00			2,000		
1990						44.00	3.00			2,000		
1991						44.00	3.00			2,000		
1992						44.00	3.00			2,000		
1993						44.00	3.00			2,000		
1994						44.00	3.00			2,000		
1995						44.00	3.00			2,000		
1996						44.00	3.00			2,000		
1997						44.00	3.00			2,000		
1998						44.00	3.00			2,000		
1999						44.00	3.00			2,000		
2000						44.00	3.00			2,000		
2001						44.00	3.00			2,000		
2002						44.00	3.00			2,000		
2003						44.00	3.00			2,000		


SIGNING – ON
Active Workspace
TYING OF FUNCTIONS
First File
FUNCTIONS OF THE MODEL
READING (OR CREATION) OF A CASE
Second File
DATA[1]
CHANGES ON SCALARS OR VECTORS[2]
SELECTION OF ONE TAX FUNCTION ON FILE
ALL CALCULATIONS
CHOICE OF REPORTS
PRINTING[3] OF REPORTS TO LOCAL OR REMOTE LOCATIONS
NEW CASE STUDY OR SIGNING – OFF

NOTE: [1] CASES ARE SECURED; READ/WRITE ACCESS PERMISSION IS RESTRICTED.
[2] THESE CHANGES CANNOT BE SAVED UNLESS WRITE ACCESS IS PERMITTED.
[3] OPTIONALLY NORMAL OR HIGH-SPEED PRINTING.

COALMOD System Overview

Figure 1



NOTE: [1] USING PHYSICAL PARAMETERS (STRIPPING RATIO, CLEAN COAL YIELD, SEAM THICKNESS, ETC.)
[2] WORKING CAPITAL CAN BE CALCULATED BY DEFAULT EQUATION.

COALMOD Calculations Overview

Figure 2

THE INDUCTIVE METHOD OF INTRODUCING APL

Kenneth E. Iverson
I.P. Sharp Associates
Toronto, Ontario

Because APL is a language, there are, in the teaching of it, many analogies with the teaching of natural languages. Because APL is a **formal** language, there are also many differences, yet the analogies prove useful in suggesting appropriate objectives and techniques in teaching APL.

For example, adults learning a language already know a native language, and the initial objective is to learn to translate a narrow range of thoughts (concerning immediate needs such as the ordering of food) from the native language in which they are conceived, into the **target** language being learned. Attention is therefore directed to imparting effective use of a small number of words and constructs, and not to the memorization of a large vocabulary. Similarly, a student of APL normally knows the terminology and procedures of some area of potential application of computers, and the inital objective should be to learn enough to translate these procedures into APL. Obvious as this may seem, introductory courses in APL (and in other programming languages as well) often lack such a focus, and concentrate instead on exposing the student to as much of the vocabulary (i.e., the primitive functions) of APL as possible.

This paper treats some of the lessons to be drawn from analogies with the teaching of natural languages (with emphasis on the inductive method of teaching), examines details of their application in the development of a three-day introductory course in APL, and reports some results of use of the course. Implications for more advanced courses are also discussed briefly.

1. The Inductive Method

Grammars present general rules, such as for the conjugation of verbs, which the student learns to apply (by **deduction**) to particular cases as the need arises. This form of presentation contrasts sharply with the way the mother tongue is learned from repeated use of particular instances, and from the more or less conscious formulation (by **induction**) of rules which summarize the particular cases.

The inductive method is now widely used in the teaching of natural languages. One of the better-known methods is that pioneered by Berlitz [1] and now known as the "direct" method. A concise and readable presentation and analysis of the direct method may be found in Diller [2].

A class in the purely inductive mode is conducted entirely in the target language, with no use of the student's mother tongue. Expressions are first learned by imitation, and concepts are imparted by such devices as pointing, pictures, and pantomime; students answer questions, learn to ask questions, and experiment with their own statements, all with constant and immediate reaction from the teacher in the form of correction, drill, and praise, expressed, of course, in the target language.

In the analogous conduct of an APL course, each student (or, preferably, each student pair) is provided with an APL terminal, and with a series of printed sessions which give explicit expressions to be "imitated" by entering them on the terminal, which suggest ideas for experimentation, and which pose problems for which the student must formulate and enter appropriate expressions. Part of such a session is shown as an example in Figure 1.

SESSION 1: NAMES AND EXPRESSIONS

The left side of each page provides examples to be entered on the keyboard, and the right side provides comments on them. Each expression entered must be followed by striking the RETURN key to signal the APL system to execute the expression.

```
      AREA←8×2                         The name AREA is assigned to the result
      HEIGHT←3                         of the multiplication, that is 16
      VOLUME←HEIGHT×AREA
      HEIGHT×AREA                      If no name is assigned to the result, it
48                                     is printed
      VOLUME
48
      3×8×2
48
      LENGTH←8 7 6 5                   Names may be assigned to lists
      WIDTH←2 3 4 5
      LENGTH×WIDTH
16 21 24 25
      PERIMETER←2×(LENGTH+WIDTH)  Parentheses specify the order in which
      PERIMETER                        parts of an expression are to be
20 20 20 20                            executed
      1.12×1.12×1.12                   Decimal numbers may be used
1.404928
      1.12*3                           Yield of 12 percent for 3 years
1.404928
```

SAMPLE PORTION OF SESSION

Figure 1

Because APL is a formal "imperative" language, the APL system can execute any expression entered on the terminal, and therefore provides most of the reaction required from a teacher. The role of the instructor is therefore reduced to that of tutor, providing explicit help in the event of severe difficulties (such as failure of the terminal), and general discussion as required. As compared to the case of a natural language, the student is expected, and is better able, to assess his own performance.

Applied to natural languages, the inductive method offers a number of important advantages:

1. Many dull but essential details (such as pronunciation) required at the outset are

acquired in the course of doing more interesting things, and without explicit drill in them.

2. The fun of constantly looking for the patterns or rules into which examples can be fitted provides a stimulation lacking in the explicit memorization of rules, and the repeated examples provide, as always, the best mnemonic basis for remembering general rules.

3. The experience of committing error after error, seeing that they produce no lasting harm, and seeing them corrected through conversation, gives the student a confidence and a willingness to try that is difficult to impart by more formal methods.

4. The teacher need not be expert in two languages, but only in the target language.

Analogous advantages are found in the teaching of APL:

1. Details of the terminal keyboard are absorbed gradually while doing interesting things from the very outset.

2. Most of the syntactic rules, and the extension of functions to arrays, can be quickly gleaned from examples such as those presented in Figure 1.

3. The student soon sees that most errors are harmless, that the nature of most are obvious from the simple error messages, and that any adverse effects (such as an open quote) are easily rectified by consulting a manual or a tutor.

4. The tutor need only know APL, and does not need to be expert in areas such as financial management or engineering to which students wish to apply APL, and need not be experienced in lecturing.

2. The Use Of Reference Material

In the pure use of the inductive method, the use of reference material such as grammars and dictionaries would be forbidden. Indeed, their use is sometimes discouraged because the conscious application of grammatical rules and the conscious pronunciation of words from visualization of their spellings promotes uneven delivery. However, if a student is to become independent and capable of further study on his own, he must be introduced to appropriate reference material.

Effective use of reference material requires some practice, and the student should therefore be introduced to it early. Moreover, he should not be confined to a single reference; at the outset, a comprehensive dictionary is too awkward and confusing, but a concise dictionary will soon be found to be too limited.

In the analogous case of APL, the role of both grammar and dictionary is played by the reference manual. A concise manual limited to the core language [3] should be supplemented by a more conprehensive manual (such as Berry [4]) which covers all aspects of the particular system in use. Moreover, the student should be led immediately to locate the two or three main summary tables in the manual, and should be prodded into constant use of the manual by explicit questions (such as "what is the name of the function denoted by the comma"), and by glimpses of interesting functions.

3. Order Of Presentation

Because the student is constantly striving to impose a structure upon the examples presented to him, the order of presentation of concepts is crucial, and must be carefully planned. For example, use of the present tense should be well established before other tenses and moods are introduced. The care taken with the order of presentation should, however, be unobtrusive, and the student may become aware of it only after gaining experience beyond the course, if at all.

We will address two particular difficulties with the order of presentation, and exemplify their solutions in the context of APL. The first is that certain expressions are too complex to be treated properly in detail at the point where they are first useful. These can be handled as "useful expressions" and will be discussed in a separate section.

The second difficulty is that certain important notions are rendered complex by the many guises in which they appear. The general approach to such problems is to present the essential notion early, and return to it again and again at intervals to reinforce it and to add the treatment of further aspects.

For example, because students often find difficulty with the notion of literals (i.e., character arrays), its treatment in APL is often deferred, even though this deferral also makes it necessary to defer important practical notions such as the production of reports. In the present approach, the essential notion is introduced early, in the manner shown in Figure 2. Literals are then returned to in several contexts: in the representation of function definitions; in discussion of literal digits and the functions (⍕ and ⍎) which are used to transform between them and numbers in the production of reports; and in their use with indexing to produce barcharts.

Function definition is another important idea whose treatment is often deferred because of its seeming complexity. However, this complexity inheres not in the notion itself, but in the mechanics of the general del form of definition usually employed. This complexity includes a new mode of keyboard entry with its own set of error messages, a set of rules for function headers, confusion due to side-effects resulting from failure to localize names used or to definitions which print results but have no explicit results, and the matter of suspended functions.

SESSION 4: LITERALS

```
      JANET←5                     Janet received 5 letters today
      MARY←8

      MARY⌈JANET                  The maximum received by one of them
8
      MARY⌊JANET                  The minimum
5
      MARY>JANET                  Mary received more than Janet
1
      MARY=JANET                  They did not receive an equal number
0
```

What sense can you make of the following sentences:

JANET has 5 letters and *MARY* has 8

JANET has 5 letters and *MARY* has 4

'*JANET*' has 5 letters and '*MARY*' has 4

The last sentence above uses quotation marks in the usual way to make a **literal reference** to the (letters in the) name itself as opposed to what it denotes. The second points up the potential ambiguity which is resolved by quote marks.

```
      LIST←24.6 3 17
      ρLIST
3
      WORD←'LIST'
      ρWORD
4
      SENTENCE←'LIST THE NET GAINS'
```

INTRODUCTION OF LITERALS

Figure 2

All of this is avoided by representing each function definition by a character vector in the direct form of definition [5 6]. For example, a student first uses the function *ROUND* provided in a workspace, then shows its definition, and then defines an equivalent function called *R* as follows:

```
      ROUND 24.78 31.15 28.59
25 31 29

      SHOW 'ROUND'
ROUND:⌊.5+W

      DEFINE 'R:⌊.5+W'

      R 24.78 31.15
 25 31
```

The function *DEFINE* compiles the definition provided by its argument into an appropriate del form, localizes any names which appear to the left of assignment arrows in the definition, provides a "trap" or "lock" appropriate to the particular APL system so that the function defined behaves like a primitive and cannot be suspended, and appends the original argument in a comment line for use by the function *SHOW*.

This approach makes it possible to introduce simple function definition very early and to use it in a variety of interesting contexts before introducing conditional and recursive definitions (also in the direct form), and the more difficult del form.

4. Teaching Reading

It is usually much easier to **read** and comprehend a sentence than it is to **write** a sentence expressing the same thought. Inductive teaching makes much use of such reading, and the student is encouraged to scan an entire passage, using pictures, context, and other clues, to grasp the overall theme before invoking the use of a dictionary to clarify details.

Because the entry of an APL expression on a terminal immediately yields the overall result for examination by the student, this approach is particularly effective in teaching APL. For example, if the student's workspace has a table of names of countries, and a table of oil imports by year by country by month, then the sequence:

```
      N←25
      B←+/[1]+/[3] OIL

      COUNTRIES,'.⎕'[1+B∘.≥(⌈/B)×(⍳N)÷N]
```

produces the following result, which has the obvious interpretation as a barchart of oil imports:

```
ARABIA   ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕.....
NIGERIA  ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕.......
CANADA   ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕..........
INDONESIA⎕⎕⎕⎕⎕⎕⎕⎕⎕................
IRAN     ⎕⎕⎕⎕⎕⎕⎕⎕.................
LIBYA    ⎕⎕⎕⎕⎕⎕⎕⎕.................
ALGERIA  ⎕⎕⎕⎕⎕⎕⎕⎕.................
OTHER    ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
```

Moreover, because the simple syntax makes it easy to determine the exact sequence in which the parts of the sentence are executed, a detailed understanding of the expression can be gained by executing it piece-by-piece, as illustrated in Figure 3. Finally, such critical reading of an expression can lead the student to formulate his own definition of a useful related function as follows:

```
      DEFINE ⍞
BARCHART:'.⎕'[1+ω∘.≥((⍳α)÷α)×⌈/ω]
```

5. Useful Expressions

As remarked in Section 3, some expressions are too useful and important to be deferred to the point that would be dictated by the complexity of their structure. In APL such expressions can be handled by introducing them as defined functions whose use may be grasped immediately, but whose internal definition may be left for later study.

For example, files can be introduced in terms of the functions *GET*, *TO*, *RANGE*, and *REMOVE*, illustrated in Figure 4. These can be grasped and used effectively by the student at an earlier stage and with much greater ease than can the underlying language elements from which they must be constructed in most APL systems.

A further example is provided by the function needed to compile, display, and edit the character vectors used in direct definition of functions. For example, an editing function which deletes each position indicated by a slash, and inserts ahead of the position of the first comma any text which follows it (in the manner provided for del editing in many APL systems) is illustrated in Figure 5.

```
      N                              The width of the barchart
25
      Q←(⍳N)÷N                       Numbers from 0 to 1 in 25 equal steps
                                          (display if desired)
      ⌈/B                            The largest value to be charted
      C←(⌈/B)×Q                      Numbers from 0 to the largest value to be
                                     charted

      S←B∘.≥C                        Comparison of each value of B with
      S                              each value in the range to be charted
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

      3 21↑1+S                       Examine a piece of 1+S
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1
```

DETAILED EXECUTION OF AN EXPRESSION

Figure 3

If the first dimension of an array (list, table, or list of tables) has the value *N*, (for example, 1↑ρ*OIL* is 7), then it may be distributed to *N* items of a file by a single operation. For example:

```
      OIL TO 'IMPORTS 72 73 74 75 76 77 78'
```

*Use the function *GET* to retrieve individual items from the *IMPORTS* file to verify the effect of the preceding expression.

```
      COUNTRIES TO 'IMPORTS 1'Non-numeric data may be entered
```

The functions *RANGE* and *REMOVE* are useful in managing files:

```
      RANGE 'IMPORTS'            Gives range of indices
1 72 73 74 75 76 77 78'

      REMOVE 'IMPORTS 73 75 77'  Removes odd years

      RANGE 'IMPORTS'
1 72 74 76 78
```

FUNCTIONS FOR USING FILES

Figure 4

Deferral of the internal details of the definition of these essential functions can, in fact, be turned to advantage, because they provide interesting exercises in reading (using the techniques of Section 4) the definitions of functions whose purposes are already clear from repeated use. For example, critical reading of the following definition of the function *EDIT* is very helpful in grasping the important idea of recursive definition:

```
EDIT:EDIT(A DELETE K↑ω),(1↓K↓A),(K←+/∧\A≠',')↓ω:0=ρA←⍞,0ρ⎕←ω:ω

DELETE:(~(ρω)↑'/'=α)/ω
```

Analysis of the complete set of functions provided for the compilation from direct definition form also provides an interesting exercise in reading, but one which would not be completed, or perhaps even attempted, until after completion of an introductory course. Extensive leads to other interesting reading, of both workspaces and published material, should be given the student to encourage further growth after the conclusion of formal course work.

```
      TEXT←'DDELLLETN AND INSRTION'
      Z←EDIT TEXT                   Apply EDIT to erroneous text
DDELLLETN AND INSRTION                     Line printed by the function
/   //   ,IO                        Line entered on keyboard
DELETION AND INSRTION                      Line printed by the function
                    ,E              Line entered on keyboard
DELETION AND INSERTION                     Line printed by the function
                                    Empty line entered on keyboard (carriage
                                    return alone) ends execution of EDIT


      DEFINE 'REVISE:DEFINE EDIT SHOW ω'   Define a function for revision

      REVISE 'SUM'
SUM:+/[α]ω
///,MAX
MAX:+/[α]ω
    /,⌈
MAX:⌈/[α]ω
```

FUNCTIONS FOR EDITING AND REVISION

Figure 5

Advanced Courses

Advanced language courses can also employ the inductive method, but the greater the student's mastery of a language, the greater the potential benefits of the deductive approach and of explicit analysis of the structure of the language. A point sometimes made in the advanced treatment of natural languages is that grammar and related matters can now be discussed **in the target language**, avoiding distractions and distortions which might be introduced by use of the mother tongue.

Similar remarks apply to advanced APL courses. In particular, the use of APL in its own discussion and in the introduction of the more complex functions is quite productive. For example, reduction is very useful in discussing the inner product, and inner product and grade are helpful in analyzing dyadic transpose.

Conduct Of The Course

The introductory course on which these remarks are based evolved through four versions offered over a period of several months. The resulting course covers three contiguous days, and has been offered a number of times in the final form.

Most students appear to work better in pairs than when assigned individually to terminals. Because there are no lectures, each pair can work at their own pace. Observations and student comments show that they find it more stimulating than a lecture course, and tend to come early and work late. Moreover, they learn to consult manuals much more than in a lecture course, and exhibit a good deal of independence by the end of the three days.

Acknowledgements

I am indebted to a number of my colleagues at I.P. Sharp Associates, to Mr. Roland Pesch for his development of the file functions used, to Mr. Pesch and Mr. Michael Berry for assistance and advice in the early versions of the course, and to Ms. Nancy Neilson and Dr. Paul Berry for comments arising from their use of the course. I am also indebted to several members of the staff of the Berlitz School of Languages in Toronto, to Ms. Grace Palumbo and Ms. G. Dunn for discussions of the direct method, and to Ms. N. Eracleous for patient demonstrations of it.

References

[1] Berlitz, M.D., **Methode Berlitz**, Berlitz and Co., New York, 1887.

[2] Diller, Karl Conrad, **The Language Teaching Controversy**, Newbury House Publishers, Inc., Rowley, Massachusetts, 1978.

[3] **APL Language**, IBM Form #GC26-3847, IBM Corporation.

[4] Berry, P.C., **SHARP APL Reference Manual**, I.P. Sharp Associates.

HOW THE PACKAGE DATA-TYPE HAS AFFECTED PROGRAMMING

Paul Berry
I.P. Sharp Associates, Inc.
Palo Alto, California

Introduction

APL has always had two ways of referring to things: by **name** and by **position**. Within an array, a sub-array is referred to by position. But within a workspace, objects are referred to by name.

APL is an array-oriented language. When you use a scalar dyadic function, the elements of one array are matched by position with the corresponding elements of the other. But in a workspace, objects are identified only by name. Even though execution takes place in an environment called a work**space**, you get no clues to that space's geography, or even that it has spatial limits (except from such occasional reminders as the message *WS FULL*).

In 1977, SHARP APL introduced a new data type, and gave it the name "package". A package is a single data object whose internal structure is like a workspace rather than like an array. Package became an alternative to array; until then array had been the only data structure in APL.

Packages were intended more as a convenience than as a theoretical extension of the APL language. Their introduction has had little effect on the algorithms that it is possible or convenient to express in APL; nor was it intended to. But it has had a considerable impact on the way programs, applications, and workspaces are organized. In this talk, I'll first review the attributes of packages, and then describe some of the practical consequences of using them.

Contrast between "array" and "package"

A package differs from an array in two main ways:

1. The variety of objects it may contain.

2. How the contained objects are arranged (and thus how you select the particular ones you need).

To summarize what's different about a package, consider the following list of attributes, first as they apply to an array and then as they apply to a package:

ARRAY

Items within an array: Each item within an array is a **scalar**.

a. A scalar has no visible structure.

(At present, it has no invisible structure either: it's just a single number or character. Eventually that restriction will be relaxed. I'll discuss that possibility in the section headed "Packages vs. extension to arrays".)

b. A scalar within an array has no name.

(The array as a whole may have a name, but not a scalar within it.)

c. All the scalars within an array must be of the same type; that is, they must all be numeric or all be characters.

This restriction is more for the convenience of the implementation than inherent in the APL language, and is likely to disappear eventually.

d. Within a numeric array, all the scalars must be of the same internal type.

This restriction is solely for the convenience of the implementation. Conversion between internal types is automatic, and in principle should be invisible to the user. However, internal type has major practical consequences for the amount of space needed to store an array. The presence of even one element that requires a more space-consuming representation requires the system to represent all elements that way, although they might otherwise have been represented by integer or Boolean forms.

Structure of an array: The scalars within an array are arranged in a rectangular grid having any number of dimensions or **axes**. The number of axes an array has is called its **rank**. You can inquire about the shape of an array variable *V* by the expression ⍴*V*, which reports the length of each axis, and hence about its rank by ⍴⍴*V*.

Selecting part of an array: It's possible to form a new array by selecting some of the elements from an array. The scalars to be selected are identified by their positions (that is, **indexes**) on the various axes.

PACKAGE

Items within a package: A package consists of a list of **names** together with the **referent** of each. A name in a package may refer to:

a. A variable.

The variable may have any structure (package or array), and type (character or numeric), or any internal representation (Boolean, integer, or floating point).

b. A defined function.

A defined function in a package has the same internal representation as in a workspace; it is **not** represented by the array of characters generated by

⎕*CR* or by the ∇ editor. Retaining the internal representation reduces the cost of moving a function definition into or out of a package. It also permits a package to contain a locked function.

c. Nothing at all.

When a name with no referent is brought into the workspace from a package, the effect is the same as expunging that name. (The fact that a package may contain a name with no referent is what makes it preferable to describe a package as "a list of names, together with their referents": there may be more names than objects referred to.)

Structure of a package: A package has no spatial or geographic structure. Items within it are referred to solely by name. If the variable *V* is a package, ⎕*PNAMES V* reports the names it contains, but ⍴*V* is a domain error.

Selecting part of a package: One or more of the objects within a package can be selected by name to form another package containing the selected names and their referents.

Alternatively, you can cause selected names from the package to appear in the workspace with the function ⎕*PDEF*. Its left argument specifies which names you want; if you omit the left argument, you get them all. Because the various names have referents which cannot be combined in a single result, ⎕*PDEF* has no explicit result. Instead, each of the names and its referent materializes in the environment outside the package as a side-effect. The expression *'X'* ⎕*PVAL PKG* gives you the value of *X* in package *PKG*, but you can only use it for one name at a time, and that must be the name of a variable.

We don't seem to have an agreed-on English word that means "extract the referent of a name from a package and materialize it in the workspace". Since this is usually done with ⎕*PDEF*, one is tempted to say "define", but that risks confusion with function definition (either with ⎕*FX* or with the ∇ function editor), or with the ordinary English use in the sense of "defining terms". So I shall say "extract" even though that isn't an accepted APL term.

There's nothing you can do with an object in a package other than store it (as part of the package) or extract it. Only after the packed object has been brought into the general workspace environment can you make use of it.

Characteristics of packages relevant to programming

The introduction of packages has the following practical consequences for the design of applications:

1. Disparate data (which previously would have required separate arrays) can be stored together in a single package.

2. Functions can be stored in a package:

a. without conversion to and from character representations;

b. many in the same package (and in the same package with variables).

3. Objects within a package are indentified by name.

4. Each package is self-contained; the names within it are unrelated to whatever uses they may have in another package, or in the environment outside the package.

In this talk, I'll illustrate these effects (together with some of the benefits and problems that grow out of them) and speculate on their more general effects on the organization of applications. Before turning to those specifics, consider two more general points: how a package resembles a workspace, and how a package resembles (and makes obsolete) a group.

Package and workspace

A package is a variable organized more like a workspace than like an array. (Indeed, at one point during development, the as-yet-unnamed structure was known informally as a "mini-WS".)

Just as the names used in one workspace are quite independent of those used in another, so the names used inside a package are independent of those in the workspace that contains the package, or of other packages in the workspace, or even of other packages inside a given package. This has two consequences:

1. You can store in the workspace different objects with the same name.

2. When you extract a named object from a package, its name may already be in use.

Since a package, like a workspace, may contain both variables and functions, it's possible to construct a package containing all the functions and variables visible in a workspace. An expression such as *⎕PACK ⎕NL* 2 3 forms a single package containing all of the user-defined functions and variables in a workspace. If you write that package to a file, you have done something analogous to saving a workspace. Your package is a single variable containing the name of every function or variable visible in the workspace, together with the referent of each. (It's possible to include system variables in a package, but since their names don't appear in the list produced by *⎕NL* 2, you have to mention them explicitly.)

A package does not contain a state indicator, so putting all the functions and variables into a package is like executing *)SAVE* at a time when no functions are active. Moreover, if there are functions on the state indicator when you form the package, some of the names reported by *⎕NL* 2 may be local names, in which case the referents packed in your package will be the local referents of those names.

APL\360 contained a mechanism for bringing objects to the active workspace from a saved workspace. That was (and still is) the system command *)COPY*. The command *)COPY* is followed by the name of a saved workspace, and then (optionally) by the names of the objects to be copied. The function *⎕PDEF* provides a similar service, with a similar option to extract either the entire namelist from the package, or just the items specified. But where *)COPY* transfers objects from a saved workspace (**outside** the active workspace), *⎕PDEF* extracts them from a package (**inside** the active workspace).

Package and group

In APL\360, a mechanism was provided to make it easier to copy and erase. By entering from the terminal a command such as

```
        )GROUP DATAGRP A B C
```

you could form a group called *DATAGRP*. The group thus formed was simply a namelist, and did not itself contain the referents of the names (and so it required no additional space for storing the referents, and permitted the membership of groups to overlap). When you subsequently entered the command

```
        )ERASE DATAGRP
```

the system erased any global objects named in *DATAGRP*'s namelist. Similarly, if the group *DATAGRP* existed in a saved workspace called *WS*, entering from the keyboard the command

```
        )COPY WS DATAGRP
```

caused the system to bring into your active workspace any global objects which were saved there and appeared in *DATAGRP*'s namelist.

The concept of group and its accompanying mechanism has largely been abandoned. In the Sharp system, the group facility still exists, but is considered obsolete. The disadvantages of groups were many. The commands that referred to groups could only be used when entered from the keyboard during immediate execution. A group could not appear in an APL expression, and so could neither be formed nor used by a function. The membership list of a group referred only to global objects. Although the name of a group couldn't appear in an APL expression, it nevertheless existed in the workspace (and so might conflict with global use of that name to refer to a function or variable). The mechanism for forming, amending, or terminating group definitions was awkward.

But despite all these shortcomings, groups did provide an important service: they permitted you to copy together a collection of disparate objects. The need for groups did not really diminish until packages provided a more powerful mechanism. Moreover, packages aren't a total replacement for groups since the entire package (which includes the referents of names) must be present in the workspace when objects are extracted from it, whereas that wasn't necessary in order to copy a group.

Keeping together related but disparate data

A package permits you to store as a single object a collection of arrays of different shape or type. Consider MABRA, for example. MABRA is a system for storing rectangular records. There can be any number of records, each described by its value on each of the **fields** that characterize the particular data base. (A MABRA data base is said to be "rectangular" because every record has the same fields.)

A particular MABRA data base can have any number of fields, each of any width or type. The creator specifies what fields exist, what each field is called, and what shape and type of data may appear in it. The MABRA command *LF* lists the fields; in an application describing personnel data, the output produced by *LF* might include the following (omitting some other parts of the display that aren't relevant here):

```
    LF
* LIST FIELDS

NAME            TYPE WIDTH
INITIALS        CHAR    3
SURNAME         CHAR   15
BIRTH           DATE    1
SEX             CHAR    1
NUMBER          CHAR    9
ACTIVE          BOOL    1
JOBCODE         INT     1
HIREDATE        DATE    1
JOBTITLE        CHAR   25
GRADE           INT     1
LOCATION        CHAR    4
SALARY          REAL    1
```

As a MABRA user, you can display any combination of fields in any sequence; you don't see (and don't need to see) how the various data arrays are stored. Clearly many different arrangements of storage are possible. Before the introduction of packages, probably the data of various types would have been stored in different file components; it would have been up to the application to maintain a directory showing where the various fields were located.

In fact, MABRA stores all the fields together, using packages to permit the different data types to share the same component. (An "inverted" field also has an additional copy of its data, stored apart from the data of other fields.) If you examine the file in which the MABRA system keeps its data, you'll find each component is a package. Each package contains the same names; the names are derived from the names of the fields. Suppose you've brought into the workspace one such package from the personnel application just mentioned, and made it the variable *P*. The names within the package become visible thus:

```
      ,' ',⎕PNAMES P
 ⍙INITIALS ⍙SURNAME  ⍙BIRTH    ⍙SEX      ⍙NUMBER   ⍙ACTIVE
 ⍙JOBCODE  ⍙HIREDATE ⍙JOBTITLE ⍙GRADE    ⍙LOCATION ⍙SALARY
```

If you were to extract the value of each of those variables, you'd find that each is a matrix. All the matrices have the same number of rows (for example, 50), and as many columns as are indicated by the parameter "width" for that field. When you use the workspace *MABRAUTIL* rather than *MABRA*, the system returns the data you requested as a set of named variables, much as it stores them internally within each package.

The fact that data arrays for the various fields are contained in separate variables makes it convenient for MABRA to use *⎕FMT* to generate displays. The formatting primitive *⎕FMT* is a misfit in APL because of its eccentric syntax. Its right argument may be a list of the names of matrices to be formatted (rather than a single array). However, a MABRA package contains a collection of matrices, each identified by name. MABRA forms a character vector containing the names of the needed arrays (set off from each other by semicolons). When that vector is catenated to the symbols '*⎕FMT*' and an appropriate left argument, the statement can be executed with ⍎ . MABRA maintains a package which contains the *⎕FMT* format phrase needed to display each field.

Data compression for sparse arrays

The ability to store together data arrays of different type and shape is exploited in some schemes for storing sparse arrays. In applications in such diverse fields as architecture, accounting, circuit design, economic forecasting, or text-processing, it is convenient to carry out computation on matrices, even though many of the elements have the value zero or blank. (There are also applications in which data may be conceived as sparse matrices, but in which it's inefficient or inconvenient to compute with them in that form. But those aren't the ones I mean here.) If the work requires storing large quantities of such data, it may save space (and hence money) to "compress" them for storage, and expand them to their full rectangular shape only when working with them.

To reduce the storage costs you might construct a Boolean array with a `1` to mark each position where there is non-zero (or non-blank) data. Then you can keep the actual non-zero (or non-blank) values in a much smaller array. This requires two arrays which differ both in type and in shape. You can keep them together by putting them into a package. The functions *COMPRESS* and *EXPAND* illustrate the compression technique method used for storage in the aviation data base ER586 (described by Dave Keith in the Sharp Newsletter for September/October 1978). In that data base, storage for 10 million sets of time series data as full matrices would have required 3110 million bytes, but the compressed form takes only 500 million.

The function *COMPRESS* takes a sparse array in conventional rectangular form, and produces a package, while the function *EXPAND* does the reverse:

```
     ∇  PKG←COMPRESS ARRAY; MASK
[1]     MASK←ARRAY≠1↑0ρARRAY
[2]     ARRAY←(,MASK)/,ARRAY
[3]     PKG←⎕PACK 'MASK ARRAY'
     ∇

     ∇  ARRAY←EXPAND PKG; MASK
[1]     'MASK ARRAY' ⎕PDEF PKG
[2]     ARRAY←(ρMASK)ρ(,MASK)\ARRAY
     ∇
```

The example shown here deals only with the elimination of blanks or zeros; clearly, the underlying technique could be extended to other "fill" characters, or to repetitions of various kinds.

Passing a single argument, returning a single result

A major advantage of work with APL is the ease with which you can create modular structures. A task is readily divided into sub-functions, each receiving as its argument the data it needs, and returning a result which can then be passed to another function. The overall structure of a set of programs is clearest (and most easily understood and maintained) when each module has only clear and explicit links to the others. Design is simplest when each defined function behaves as the primitive functions do: it receives no information other than the arguments explicitly passed to it, and sends no information other than its explicit result. (You might use the word "pseudo-primitive" to describe a defined function which behaves in the same way as a primitive.)

Organizing work into discrete modules is partly a matter of choosing an appropriate way to represent and think about the problem. But it is also depends in part on having ways to keep related data together. There are some "arguments" which can't be conveniently represented in a single array, or even in a pair of arrays. Likewise, there are some results which, even if you think of them as a single phenomenon, can't be represented in a single array. Combining arguments or results into a package provides a way in which clean organization can be preserved, although at the expense of having to pack and unpack the arrays that are passed together.

As an example, consider a workspace for manipulating lists. A "list" is not a recognized data structure in APL. I'm using the term to describe what is in effect a vector of vectors. It's a form in which you might want to store items of text having irregular and unpredictable length. Suppose, for example, in a data base on publications, you have a field for the abstract of each publication. A publication's abstract may be a single sentence, a substantial paragraph, or even several pages — or there may be none at all. (Representing the abstracts is a variant of the data compression problem.)

You might elect to keep this data as a list. A list could be represented by a vector containing all of the text (but no extra blanks), together with a table of pointers showing the length or perhaps the amount of offset before each item. (Both length and offset will be needed to select or search for items; whether you store both explicitly, or recalculate them as needed, may depend on the trade-off between the costs of space to store a more extensive representation, and the CPU time needed to calculate positions repeatedly.)

A workspace could be constructed which handles lists as single objects. (There's a sample on the Sharp Toronto system in workspace 880 *LISTS*.) The central assumption of such a workspace is that a list is a single APL variable which behaves like a vector of vectors. If *L* is a list, you can select from it (for example) its 4th and 17th members by an expression such as

```
      L SUB 4 17
```

This returns a matrix with two rows and as many columns as are needed to accomodate whatever turns out to be contained in items 4 and 17. (The function *SUB* might have been defined to return a sub-list of just the members you requested, with a second function available to convert that list into a matrix if desired.)

Since list is not really a primitive structure in APL, the workspace provides the function *ENLIST* and *DELIST* to convert to and from conventional matrix forms (with trailing blanks or zeros). These use the utility functions *RAVEL* and its inverse *MATRIXFORM*. Given a matrix and a vector showing how many characters are wanted from each row, *RAVEL* forms a vector from which the remaining characters have been excluded. This is the basis of the internal storage of a list: a vector (here called *LST*) containing the characters in each item but nothing extra, and a numeric vector (here called *IX*) containing the length of each item. This implies that

```
      (⍴LST) ↔ +/IX
```

and the offset from the beginning of *LST* to the point at which each item starts is ¯1↓0,+\IX.

Based on those assumptions about the internal representation of a list, the functions *ENLIST* and *DELIST* can be defined as follows:

```
      ∇  Z←ENLIST X; LST; IX
[1]      X←MATRIX X
[2]      IX←+/∨\⌽X≠1↑0⍴X
[3]      LST←IX RAVEL X
[4]      Z←⎕PACK 'LST IX'
      ∇

      ∇  Z←DELIST X; LST; IX
[1]      'LST IX' ⎕PDEF X
[2]      Z←IX MATRIXFORM LST
      ∇
```

Here (as in most applications that use packages) the definitions rest upon a convention regarding the names that the package will contain and the significance of each. The function *ENLIST* creates a package containing the names *IX* and *LST*, and the function *DELIST* (like most of the other functions in the workspace) assumes that, when it receives a package argument, the argument will contain the names *LST* and *IX*. If this application used global variables rather than packages, it would need a similar convention regarding names. But when such variables are enclosed in a package, it becomes possible to have present in the workspace at the same time any number of **different** packages containing the **same** names (each with its own *LST* and *IX*) without conflict between them.

To replace an item within a list with a different item requires something analogous to indexed specification; you'd like to be able to say:

```
        L[17]←'NEW ENTRY 17'
```

This is in effect a function of three arguments: the list in which replacement is to take place, the position at which you want something inserted, and the value to be put there. Here again packages make it easy to combine arguments. The syntax adopted to replace the 17th item of *L* is:

```
        X REPLACES 'L' AT 17
```

in which *X* is the new data to be inserted. The function *AT* is simply a way of packing together the name of the list to be changed and the array of positions at which change is desired:

```
      ∇  Z←NAME AT LOC
[1]      Z←⎕PACK 'NAME LOC'
      ∇
```

The function *REPLACES* includes provision to extract the objects *NAME* and *LOC* from the package it receives as its right argument.

Shared variables as arguments

Functions defined in APL have two classic forms: one (described in the preceding section) in which the function receives input as an explicit argument and sends output as an explicit result, and the other in which this exchange is made not by arguments and results in the usual sense, but by one or more shared variables. The shared variables may serve both as the channel by which the function receives the values of its arguments, and as the channel by which it returns the value of its result.

An application using shared variables may require disparate objects as "arguments". For example, if you're using a shared-variable file access system, you may need to say, "Store data *X* at component *Y*". In a system without packages, the usual approach has been to have two shared variables, one for control information and the other for the actual data. (You could also adopt a protocol which alternated control and data uses of a single variable.) A typical application uses a control variable *CTL* to indicate what file component is to be written or read, and a data variable *DAT* for the data to be transferred. The variable *CTL* is interlocked, usually by `1 1 1 1 ⎕SVC 'CTL'`, while the variable *DAT* is not interlocked at all. Then the functions *READ* and *WRITE* appear as follows:

```
     ∇ Z←READ WHERE
[1]    CTL←WHERE
[2]    WHERE←CTL                 ⍝WAIT FOR AP TO SIGNAL DATA READY
[3]    Z←DAT
     ∇

     ∇ Z←WHERE WRITE WHAT
[1]    DAT←WHAT
[2]    CTL←WHERE
[3]    Z←CTL                     ⍝WAIT FOR AP TO SIGNAL WRITE COMPLETED
     ∇
```

Where the main reason for using two variables is that the data and the control information are disparate, you can avoid that difficulty by putting them both in a package. That permits you to use one shared variable instead of two. The definitions of read and write use a single shared variable (say, *SHV*) as follows:

```
     ∇ Z←READ WHERE
[1]    SHV←WHERE
[2]    Z←SHV                     ⍝WAIT FOR AP TO SET SHV
     ∇

     ∇ Z←WHERE WRITE WHAT
[1]    SHV←⎕PACK 'WHERE WHAT'
[2]    Z←SHV                     ⍝WAIT FOR AP TO SIGNAL WRITE COMPLETED
     ∇
```

Depending on the design, there may be advantages in keeping data and control information in separate variables. For example, if the application that receives the data uses (i.e. reads) the same value several times, you might handle that by setting an interlock on the control variable but not on the data variable. That way, after receiving (via the control variable) confirmation that a new value has been set, the program can use the data variable repeatedly without either saving a duplicate copy or affecting the interlock status of the control variable. Packing the two together would preclude that.

Temporary storage for disparate objects

There are many situations in which it's useful to have a convenient spot in which to store and then retrieve a variety of objects. Because a package can contain any collection of named objects — including other packages — it's a very useful device for temporary storage. An application in which this was essential is mentioned in McDonnell's talk on event handling (reprinted elsewhere in this volume). He describes the effort to define a function *RESUME* which has the same effect as →⎕*LC* but doesn't require the user to type either the branch arrow or the symbol ⎕. A method suggested by Brian Hart and several others achieves this by creating a ⎕*TRAP* global to the function *RESUME*. The trap awaits an event with an arbitrary number (say, 550), and provides a recovery expression which specifies that, when that event is encountered, the system should execute →⎕*LC*. Then the function *RESUME* signals that event by executing ⎕*SIGNAL* 550.

For the function to work appropriately, it is essential (1) that ⎕*SIGNAL* terminate execution of *RESUME*, and (2) that the ⎕*TRAP* which contains the recovery expression →⎕*LC* be **global** to the function *RESUME*.

It's easy enough for *RESUME* to set up a global ⎕*TRAP*. But what to do with the existing global ⎕*TRAP*? You need a place to store it temporarily. Any variable will do, provided it isn't local to your function *RESUME*. It can be an existing variable, or a new one whose name you invent. Suppose it's a variable called *X*. Before it establishes a new global ⎕*TRAP*, the function *RESUME* creates a package containing the former values of ⎕*TRAP* and ⎕*ER*, and assigns that to *X*.

An interesting question arises when the name *X* is already in use as the name of a global variable. What about the former value of *X*? Perhaps that too is going to be needed again. No problem! The former value of *X* can be included inside the new one. The statement becomes:

```
      X←⎕PACK 'X ⎕TRAP ⎕ER'
```

The recovery expression, in the new global ⎕*TRAP*, provides that, when event 550 is detected (and execution of *RESUME* has terminated) the system should first restore the old values of *X*, ⎕*TRAP*, and ⎕*ER*, and then should execute →⎕*LC*. Hence the value of ⎕*TRAP* set temporarily by *RESUME* is:

```
      ⎕TRAP←'∇ 550 E ⎕PDEF X ◇ →⎕LC'
```

As with the lists example, it is here possible to keep two different versions of the same name by enclosing them in packages (rather than by localization and shadowing). The *RESUME* example has the added twist that one value of *X* is stored inside another.

Alternate interior definitions

In trying to create programs that are clear, easy to read, and easy to maintain, a major technique is **subordination of irrelevant detail**. Programs constructed from modules permit a "main" program to make a general statement of the procedure by invoking modules from an appropriately chosen family of sub-functions. The task of development is often to start with the overall goal of the application, from that construct a likely set of sub-functions, and finally (using the sub-functions as a sort of intermediate-level application language) write the main definitions. In many situations, the sub-functions can remain unchanged, and can be used repeatedly throughout a variety of higher-level application programs.

But variation may also occur in the other direction. That is, a single set of high-level functions may require somewhat different definitions for the sub-functions, depending on the context. This is particularly likely when the same general structure of programs is used with different physical devices. For example, a program which analyzes data imported on tapes from other computing systems may have the same general structure for a wide range of tapes, but need different definitions to handle the different data-representation schemes it encounters. Roger Moore, architect of much of the Sharp system's data transmission facilities, tells me his workspace for maintaining the IBM 3705 (transmission control unit) must deal with binary representations that, depending on the context, are decoded as decimal numbers, hexadecimal characters, or arbitrary operation codes. His input/output function may use any of five different definitions for the encoding and decoding sub-functions.

Similarly, the authors of programs to generate graphic displays on a variety of devices may require the same over-all program structure, but quite different definitions for some of the I/O sub-functions.

A neat method to deal with problems of this type is to use a single outer program, but with alternate definitions for some of the sub-functions. Each set of alternate definitions is contained in a package. All the alternative packages contain the **same** named functions, but with **different** definitions. The appropriate package may appear as an explicit argument to the main function, or may be invoked in some other way.

Note that, to provide alternate interior definitions, it doesn't matter where the alternate packages are stored. In particular, they don't need to be outside the workspace. Thus the ability to provide alternate definitions is not the same as an overlay, which is usually motivated by the need to save space in the active workspace.

Packages and program overlays

The term "overlay" dates from an era when programmers needed to be aware of the physical locations at which their programs resided. An overlay involved re-use of storage. For example, a program not required until the later stages of a job could be read into storage previously allocated to a program used only in the early stages. In an APL workspace, programs don't have locations. But if space is a problem, you can still bring in definitions when they're needed, and erase those with which you're finished. There are several techniques for doing this. In the days of APL\360, it was common to find that a particular application was distributed among three or four different workspaces. You started out in workspace one, and when its portion of the work was finished, you receive a message something like this:

```
NOW SAVE THIS WORKSPACE BY ENTERING:   )SAVE PART1
THEN LOAD PHASE 2 BY ENTERING:         )LOAD PHASE2
THEN COPY DATA FROM PHASE1 BY:         )COPY PART1 DATAGRP
```

That sequence of commands was necessary because there was no way that the program could achieve the overlays automatically.

Somewhat later, people developed a technique for achieving the same effect in a more automatic way. One workspace wrote data to a file, and then used ⎕*LOAD* to cause the next workspace to be loaded. There, a latent expression took over, and read back the data previously sent to file. (More recently, the system variable ⎕*SP* has been used to make it easier for one workspace to pass information to its successor; ⎕*SP* and ⎕*HT* are at present the only variables that survive ⎕*LOAD*.)

Starting in about 1973, an alternative approach became possible. Instead of loading a substitute workspace, a single master program running in a single workspace could erase subordinate functions it didn't need, and bring in others as required. Where before there was a chain of workspaces, one linked to the next by *⎕LOAD*, a single workspace could progress through a series of phases, expunging part of its own contents and replacing those objects with others from outside.

At first this approach depended on the functions 3 *⎕FD* and 6 *⎕FD* (precursors of *⎕FX* and *⎕EX*). The functions not present in the workspace were stored on file as character arrays. They could be read into the workspace as data, and then "fixed" as functions.

With the introduction of packages, overlays became simpler to write and cheaper to execute. Because functions are stored in the package in their internal form, the translation inherent in *⎕FX* was avoided. Because a single package can contain many functions, it was no longer necessary to iterate, reading and then fixing a separate array for each function to be transferred. Because a function in a package can be locked, the security-conscious could avoid the risks in giving others access to its canonical representation. For these various reasons, overlaying with packages is becoming more and more widespread.

To illustrate function overlays with packages, consider an example from text-processing. The various activities involved in producing documents might be divided into the following phases:

1. Editing (entry and correction of the raw text)

2. Composing (preparation of galley proofs)

3. Paging (dividing the galley into pages, providing page numbers, footings, heading, etc.)

4. Sending (transmitting paged and formatted text to files for printing elsewhere)

5. Printing (display of paged and formatted text at an appropriate terminal)

Transition between these various phases can be controlled manually by the user, for example by executing a function called *EDITING* with definition such as the following:

```
     ∇ EDITING
[1]    PHASE←GETPHASE 1
     ∇
```

In similar fashion, the function *COMPOSING* is defined thus:

```
     ∇ COMPOSING
[1]    PHASE←GETPHASE 2
     ∇
```

(The names *COMPOSING*, *EDITING*, etc. are simply covers to make it unnecessary to remember the numbers associated with the various phases.)

The global variable *PHASE* indicates which phase's definitions are already present, so that the function *GETPHASE* need do nothing if definitions for the requested phase are already present.

```
        ∇  P←GETPHASE P
[1]       →(0=⎕NC 'PHASE')/GET  ⍝IF PHASE UNSPECIFIED, GET NEW ONE
[2]       →(PHASE=P)/0          ⍝IF PHASE ALREADY HERE, QUIT
[3] GET:  ⎕PDEF ⎕READ TEXTPKGS,P ⍝RESULT IS PHASE JUST BROUGHT IN
        ∇
```

Each of the functions which a user may directly invoke (for example *ENTER*, *CHANGE*, *MOVE*, *EDIT*, *PRINTSETUP*, and so on) contains as its first line a function such as *EDITING* or *COMPOSING*, etc., to assure that definitions appropriate to its phase are present in the workspace. (This example includes no explicit provision to erase the no-longer-used names, but achieves that by "expunging while extracting," which I'll mention in the next section.)

The strategy used in the function *GETPHASE* is preventive. That is, **before** proceeding, it verifies whether the needed phase is present. That isn't very expensive, but it does have some slight cost in repeated checks. A more automatic alternative becomes possible **if** (and the qualification is important) each name, when present, has one and only one definition. In that case, one might detect its absence with *⎕TRAP*, and use the recovery expression to invoke *GETPHASE* only when needed. There's a proposed extension to event trapping which would make that easier to do. It would identify a distinct event "undefined name" (whereas at present, use of a name that isn't defined is reported either as a *VALUE ERROR* or as a *SYNTAX ERROR*).

Expunging while extracting

A common technique for expunging no-longer-wanted material at the same time that new functions are brought into the workspace is to include in the package the names to be erased, but without referents. For example, in the text-processing example mentioned in the preceding section, each of the packages representing the various phases contains the **same** list of names, but with referents only for those needed in the particular phase.

An alternative technique is to make names wanted within a particular phase local to the main function for that phase. That has the advantage that those definitions, along with any other local objects, are automatically expunged as soon as the main function terminates. But that technique works best when there is a one-to-one correspondence between phase and function. Moreover, it assumes that one "main function" (with its attendant local functions) won't be used inside another, since that would bring more than one phase into the workspace at the same time. That would work, but might be counterproductive if the goal is to save space.

"Shell" applications (whose code isn't really in the workspace)

In an increasing number of application packages, the main body of programs isn't stored in the workspace at all. You load the workspace (or copy the main function from it) and, once the main function is invoked (by your explicit action or perhaps by the latent expression), the function ties a file belonging to the application's author, and reads from it the functions that do the real work. This way of organizing an application has been in use for a long time, but the introduction of packages has made it substantially easier to write and cheaper to execute.

The advantages of storing the main body of functions outside the workspace is that the "shell" functions in the workspace can be copied into another workspace:

- **a.** without requiring space for most of its definition until they are actually used;
- **b.** without requiring maintenance or correction as new editions of the working functions are released, bugs are fixed, and so on.

The application author is thus assured that all users of the application receive any updates or corrections. (Of course, this has also the disadvantages that all users also receive automatically any errors the author may introduce.)

There are various ways to maintain the master copies of the working functions. Usually the author maintains a private workspace in which the functions reside, and in which corrections or revisions can be made. The author's workspace includes a function which puts together the appropriate packages and stores them in the file from which the public cover-function takes them — after archiving the earlier versions.

Where there may be security or proprietary considerations, you can lock the cover function, and make the working functions local to that locked outer function. The workspace saved in the public library doesn't contain the working functions. Perhaps the outer function brings in the working definitions only for users who are authorized or enrolled. A suitable *⎕TRAP* can make it impossible for anyone to halt execution in such a way that the working functions remain visible.

Utilities built from modules vs. utilities that are easy to copy

A utility function is one which does not stand as a complete application, but may serve as a module in many applications. Usually each programmer maintains a private library of functions he or she often finds useful. Members of a community of programmers are likely to develop a common set of utilities that they all use. That greatly simplifies the task of writing new applications, and, when utilities are shared across a community of users, makes it much easier for one group to understand or maintain programs written by other members of the group.

Utility functions are used to best advantage when the application is constructed in a highly modular style. Conversely, the existence of a library of utility functions makes it far easier to develop modular applications.

But can the utilities themselves be constructed in modular fashion? That is, can they in turn employ sub-functions from a lower strata of widely used sub-utilities?

There seem to be three main ways of treating the modularity of utilities:

1. Each utility stands alone.

The advantage of a stand-alone utility is that you can copy it into a workspace confident that

- **a.** it doesn't require anything else;

b. it won't restrain or conflict with other functions in the workspace.

The disadvantage is that a stand-alone utility may be unnecessarily lengthy or complicated, having to code for itself definitions that could more easily be provided by sub-functions, and which may already exist in the workspace.

Rohan Jayasekera and Doug Keenan, the authors of the utility collection in library 880 *SAUCE*, concluded that the advantage of being able to copy a single object without need for sub-functions or fear of conflict with sub-functions outweighed the other issues, and elected to require all their utilities to stand alone.

2. A family of utilities is modular, making free use of a shared pool of common sub-functions.

Individual authors or small writing teams tend to use this approach. It permits the individual definitions to be brief, and to make repeated use of the same sub-functions in many different contexts.

The principal disadvantage is that each utility is dependent on its sub-functions, and the sub-functions it needs must be copied with it. The act of copying them is itself a nuisance (if a minor one). But it carries with it the risk that the copied sub-functions conflict with others in the workspace. This is particularly true if the utility function is copied into a workspace some of whose programs originated elsewhere, from a community (or a time) which had different conventions about sub-functions and their names.

3. Each utility has its own sub-functions, local to it (and contained in a package which is either read from file or copied at the same time that the utility is copied).

This approach is basically the same as the application "shell", but at the level of the utility program rather than the whole application. It resembles the stand-alone approach in that each utility has its own code, quite possibly duplicating code already in the workspace. Since the utility function can be defined in terms of sub-functions, it may thereby be more readable. But its sub-functions must be extract from a package (and perhaps also from a file) every time it is executed.

It seems to me unclear which of these three alternatives has the best long-run potential. I see major advantages in approach 2 (a shared pool of common utilities), but it depends on a level of agreement and collaboration that the APL community doesn't often achieve, mainly because in APL it is so easy for individuals or very small groups to "do their own thing".

If utilities must be able to stand alone, they can combine readability and central maintenance by using local modules. Packages read from files are probably the simplest way to do that. The crucial issue may become the trade-off between design simplicity and CPU cost, which in turn depends on how often you need to execute the function, and how often you need to read or modify its definition.

The "two package" system as a substitute for workspaces

It is possible to put all of the functions that represent an application into a package (or several packages, if that's preferable). A file containing the necessary package is in many respects equivalent to a saved workspace. An expression such as

```
        ⎕PDEF ⎕READ FILE,COMPONENT
```

is thus in many respects equivalent to loading (or, more properly, to copying) a saved workspace.

Before file systems were widely used, the individual who made use of a public library workspace usually loaded the workspace, used it (entering data, generating results) and then saved the entire active workspace in his or her private library. The saved workspace contained both a copy of the public library application and also the user's private data. The workspace thus had two sets of objects: those that originated in the public library, and those specific to this use or user. The first set doesn't really change, whereas the values of objects (usually variables) in the second set is continually changing.

By capitalizing on the distinction between these two sets of objects, one arrives at the "two package" style of workspace organization.

To start using such an application, you read in a package that characterizes the unchanging aspects of the application, and extract its contents. To use a particular set of data, you read a package containing values for the second set. By forming a package of the current values of names in the second set, you store a snap-shot of all that is subject to change. If you append each such package to a file, the successive components form a succession of snapshots taken at points during the use of the application. Since the saved package contains only the data that are subject to change, total storage is less than if you'd saved the whole workspace.

The following utilities *SAVE* and *RETRIEVE* illustrate the technique.

```
     ∇  Z←SAVE
[1]     Z←TM TNO,(⎕PACK NL) ⎕APPENDR TNO
     ∇
```

The variable *NL* is the list of names subject to change in the application. It is brought into the workspace as part of the package for phase 1, containing the enduring aspects of the application. The function *TM* displays *⎕RDCI* for the file component thus appended (showing who saved it, and when), and *TNO* identifies the file.

```
     ∇  Z←RETRIEVE N
[1]     ⎕PDEF ⎕READ TNO,N
[2]     'WS DATA REPLACED BY DATA RETRIEVED FROM COMPONENT ',⍕N
[3]     Z←TM TNO,LASTRETRIEVED←N
     ∇
```

The function *RETRIEVE* can call back any of the components that have been saved. At the beginning of work, the latent expression (or some equivalent mechanism stored in the first package) may automatically retrieve a particular component (for example, the one last saved).

The variable *LASTRETRIEVED* serves to keep track of which component has been brought in, perhaps as a check against accidentally overwriting one saved component with another.

Clearly the same effect could also be achieved, perhaps more economically, by having the application save data directly in a file. The "two package" technique is particularly useful in adapting older workspaces. Because they assumed that the needed data would be saved by saving the workspace, older workspaces often make use of numerous variables that are not readily adapted to conventional data storage schemes for files. The two-package technique takes a snap-shot of all of the variables subject to change, without inquiring about their organization. Moreover, by defining the function *SAVE* so that it uses *⎕APPENDR* rather than *⎕REPLACE*, you preserve a record of **each** snapshot, whereas the command *)SAVE* would preserve only the last one.

Inscrutable programs when names are materialized from packages

There was a time when you could trace through the flow of control in an APL program just by reading its definition (together with the definitions of other functions that it might invoke). You could see the definitions of all the functions invoked because they were already present in the workspace. You could spot where any variable was set simply by looking for its name to the left of an assignment arrow.

In a large application, the task of keeping track might become tedious, but not fundamentally different. The public libraries began to offer WSDOC applications, which generated tables showing for every name in the workspace where and by what it was used or set.

The addition of ⍎ made automatic documentation much more difficult. Who could tell what name might be created in the argument of execute? The introduction of packages made things even worse. When the system executes a statement such as *⎕PDEF PKG* some number of names from inside the package *PKG* are defined into the workspace. There's no way of knowing (just from studying the program definitions) what variables are thereby be created, or given new values — or even expunged. Executing *⎕PDEF PKG* could even substitute different definitions for the sub-functions that the main definition invokes.

I don't know of any sure remedy for the inscrutability that packages may introduce in a program. But there are a few palliatives. Comments explaining what objects will be extracted from a package are doubtless helpful to a human reader, although probably not to a *WSDOC* program.

Similarly, it is helpful to make explicit what names you expect to find in a package. For example, the definition of *EXPAND* shown in the section on data compression includes the statement:

```
      'MASK DATA' ⎕PDEF PKG
```

to make explicit that those are the names that *EXPAND* needs to extract from *PKG*. The monadic form *⎕PDEF PKG* (applied to an appropriate package) would work just as well, but wouldn't be as explicitly readable. Moreover, stating explicitly what names you are extracting from a package gives you a safeguard aginst inadvertently supplying a package which contains quite different names.

Using names where before you used numbers

Passing arguments by value is characteristic of APL. In all defined functions constructed as "pseudo-primitives," a function's only input is the value of its argument. Inside the definition, the name by which the argument is known depends not at all upon the argument itself, but upon the header of the function that receives it. The function doesn't know what the argument was called, or even whether it had a name.

The APL literature is replete with techniques that depend upon the position within the argument at which the various data arrays are expected. I would distinguish three levels for the significance of position within an array argument, according to the arbitrariness of the positioning:

1. Positions with fixed significance. Example: the arguments of ⊥.

2. Positions consistent with other arrays. Example: The arguments of inner product.

3. Positions quite arbitrary (consistent only with conventions of the function).

Arrays in which particular positions are arbitrarily assigned a special significance are widely used in APL. They provide a way to combine in a single argument information which has no inherent sequence. For example, a function to calculate loan payments may accept an array of proposed loans in which, along the last axis, position 1 is principal, position 2 is rate, position 3 is term, and position 4 is frequency. A function called *PAYMENT* might use a single argument *X* but devote its initial statements to taking *X* apart. (In case you're wondering, the definition that follows uses replication rather than indexing so that the argument can be of any rank).

```
      ∇  Z←PAYMENT X; PRIN; RATE; TERM; FREQ; R; R1; T
[1]      PRIN←1 0 0 0/X
[2]      RATE←0 1 0 0/X
[3]      TERM←0 0 1 0/X
[4]      FREQ←0 0 0 1/X
[5]      R1←1+R←RATE÷FREQ
[6]      T←TERM×FREQ
[7]      Z←PRIN×(R×R1⋆T)÷(R1⋆T)-1
      ∇
```

The variable *X* is never really used as a single array, but only as a vehicle from which the variables *PRIN*, *RATE*, *TERM* and *FREQ* can be extracted. It seems likely that *PAYMENT* is invoked by an expression which forms the last axis of the argument by catenating the separate arrays for principal, rate, term, and frequency.

If an array is formed only to be taken apart again, you might consider that the data equivalent of gluing. "Gluing" is a slang term used by some members of the APL community, usually with pejorative intent. It denotes the practice of forming a single statement by linking two expressions by making them the arguments of a function which itself does nothing. The linking function becomes necessary when there's no plausible way in which you can treat the result of one expression as the argument of the other. For example, the statements *A←F X* and *B←F Y* might be glued with `,0ρ` as follows:

```
      A←F X,0ρB←F Y
```

The extent to which this method of linking statements is admirable or reprehensible has been the subject of some heated discussions. As the Romans should have said, there is no ending disputes about taste. Be that as it may, it seems to me that forming an array solely for the purpose of passing it as an argument, and then dividing it back into its original constituents, is an analogous practice. Forming a package does the same thing (that is, puts together distinct arrays), but with no need for a spatial convention regarding which part of the argument is where, and without the requirement that the various arguments conform in type or shape, and without requiring you to supply placeholders for information that is missing or which you don't care to specify. Packages thus make possible what seems to me a preferable way to write the definition of a function such as *PAYMENT*:

```
    ∇  AMT←PAYMENT X; PRIN; RATE; TERM; FREQ; R; R1; T
[1]    'PRIN RATE TERM FREQ' ⎕PDEF X
[2]    R1←1+R←RATE÷FREQ
[3]    T←TERM×FREQ
[4]    AMT←PRIN×(R×R1*T)÷(R1*T)-1
    ∇
```

Searching for objects by name

In a variety of applications, you need to find where in a directory or reference table some particular sequence of characters is found. The position at which you find a match becomes a pointer to further information. For example, you might look up in a name-table to find which row contains the characters *ABR*63*A*, and then use that information to select from other tables a list of the file components that deal with it. You might have an expression such as

```
      COMPONENTS[(TABLE∧.='ABR63A')⍳1;]
```

If the entry you're looking for could be expressed as a name, there's an alternative way to get the associated information. You build the directory not by creating a character matrix which can be searched, but by putting into a package a variable for each of the names you may want to look up. You assign to that variable a value which contains the component numbers, or whatever it is you need to retrieve. If *ABR*63*A* is an entry you may want to look up, you include in the package a variable *ABR*63*A*. The task of finding the information associated with *ABR*63*A* becomes:

```
      'ABR63A' ⎕PVAL PKG
```

(In either case, you may want to cover the possibility that the name you're seeking isn't there. You could trap an index error in `COMPONENTS[(TABLE∧.=NAME)⍳1;]`, or a domain error arising from attempt to extract a non-existent name from the package.)

A directory system based on names stored in packages is used in a number of applications, most notably in the Official Airline Guide data base. In the OAG, for each existing combination of values on the key fields, a directory package contains a corresponding variable. For example, if your request requires locating components in which the access function will find data concerning flights whose origin is Los Angeles (LAX) and destination London-Heathrow (LHR), the function constructs the name *QLAXDLHR* and then extracts the value of that name from a package of origin-destination variables.

In this sort of application, the explicit look-up provided by

```
      (TABLE∧.=SAMPLE)⍳1
```

is replaced by the system's ability to find a name in the symbol table of a package (or, for that matter, in the symbol table of a workspace). The system's internal mechanisms for searching the symbol table serve instead of an APL expression. Interestingly, my rather casual tests indicate that the system can execute ∧.= and ⍳ so fast that the package method offers no appreciable cost savings with tables of moderate size (say, under 300 entries).

A function to display the contents of a package

If *V* is the name of a variable, the expression *V* causes the system to display the value of *V* — but only when *V* is an array. When *V* is a package, the system declines any attempt at display, and simply prints the message ***PACKAGE***.

There are several reasons for that. To begin with, a package contains not just values but also names, and there's no convention for displaying names-and-their-referents. There is no defined order to the objects within a package. A package contains objects that differ in type and shape, and such objects can't be joined to form a large array, and there's no convention regarding how they should be displayed separately.

Of course it's possible to write a defined function which will effectively tell you what's in a package. For example, if *P* is a package containing names *A B C F*, you might use a function *DISP* to display it, and generate output like this:

```
      DISP P
PACKAGE CONTAINING NAMES: A B D C F

A: RANK 1, SHAPE 5
1 2 3 4 5

B: RANK 2, SHAPE 2 10
ABCDEFHHIJ
KLMNOPQRST

C: RANK 0
45

D: UNDEFINED

F: FUNCTION
    ∇ Z←F X
[1]   Z←⍉ 0 100 100 ⊤X
    ∇
```

The function *DISP* examines the package's namelist, and for each name extracts the referent and displays it. There are two complications in doing that:

1. The referent may itself be a package.

2. The referent may be a function.

When the referent is a package, the usual solution is to make the display function recursive.

When the referent is a function, you can use *⎕CR* (or 1 *⎕FD* or 2 *⎕FD*) to generate an array that represents its definition. However, the argument of *⎕CR* must be the name of a function **already visible in the workspace**. It's necessary to extract the function from the package into the workspace before you can generate its character representation. But when you extract a function from a package, there's a risk that its name will conflict with a name already in use.

There is a way around that (ingenious if not exactly elegant). Using *⎕FX*, you create a temporary function whose header localizes the name of the function you want to display. In the environment of that local function, you use dyadic *⎕PDEF* to extract the function you want to display.

```
        ∇  DISP P; NL; I; ⎕IO
[1]        NL←⎕PNAMES P
[2]        ⎕SIGNAL(2≠⍴⍴NL)/11                ⍝ARGUMENT MUST BE PACKAGE
[3]        ⍞←'PACKAGE CONTAING NAMES:' ◇ ,' ',NL
[4]        ⎕IO←1 ◇ I←0
[5] TEST: →((1↑⍴NL)<I←I+1)/0
[6]        '' ◇ NL[I;] DISP1 P               ⍝DISPLAY ONE NAMED OBJECT
[7]       →TEST
        ∇
```

The function *DISP* extracts names one at a time from the package, and for each invokes the function *DISP*1:

```
        ∇  NAME DISP1 P; F; FOO; X; RNK; RHO; Q; QN; CR; NC
 [1]       NC←NAME ⎕PNC P
 [2]       ⎕SIGNAL(NC∊ 0 4)/11
 [3]       ⍞←((NAME≠' ')/NAME),': '
 [4]      →(NC= ¯1 2 3)/UND,VAR,FN
 [5] UND:  'UNDEFINED'
 [6]      →0
 [7] VAR:  X←NAME ⎕PVAL P
 [8]      →(2=⍴⍴⎕PNAMES X)/PKG
 [9]       RNK←⍴RHO←⍴X
[10]       'RANK ',(⍕RNK),(RNK>0)/', SHAPE ',⍕RHO
[11]       X
[12]      →0
[13] PKG:  DISP X
[14]      →0
[15]  FN:  'FUNCTION'
[16]       CR←⎕AV[⎕IO+156]
[17]       Q←'''' ◇ QN←Q,NAME,Q
[18]       F←'∇Z←FOO P;',NAME
[19]       F←F,CR,'[1] ',Q,NAME,Q,' ⎕PDEF P'
[20]       F←F,CR,'[2] ⎕PDEF ',Q,'Z',Q,' ⎕PACK ⎕CR ',QN,'∇'
[21]       Q←3 ⎕FD F
[22]       DSF FOO P
        ∇
```

Further ingenuity is required in defining the temporary function *FOO*. It must return a result (containing the canonical representation of the function you want to display), and the result must have a name. But suppose the name you've used for the result is the name of the function you want to display? The temporary function *FOO* (created by lines 18-21) avoids even that problem. It does it in the following way. First it extracts the function to be displayed (having made no use yet of the name of the result). Then it creates the canonical representation of that function. Now comes the tricky part; you want to assign that canonical representation to the name used for the result, but by now that name may refer to the function-to-be-displayed. The assignment arrow ← would refuse a request to create a variable with a name used to refer to a function. But ⎕*PDEF* isn't so squeamish. Instead of using ← , the canonical representation is packed under the result-name, and then extracted from its temporary package (thereby creating the needed result variable whether or not its former referent was a function). Here is the definition of the temporary function *FOO* that would be generated to display a packed function named *F*:

```
     ∇ Z←FOO P; F
[1]    'F' ⎕PDEF P
[2]    ⎕PDEF 'Z' ⎕PACK ⎕CR 'F'
     ∇
```

This nastiness with name conflicts really arises because at present a defined function cannot be directly the result of some ⎕*FX*-like operator, nor directly the argument of some ⎕*CR*-like operator. I believe that the awkwardness in displaying a packed function arises not so much because of a difficulty in the concept of package itself, but because we lack ways to treat defined functions directly.

(The package display function listed here may be found on the Sharp Toronto system in workspace 880 *PKGDISPLAY*. There's another package display function in workspace 777 *WSDOC*, developed by Leslie Goldsmith. It shows each level of packaging by progressive indentation from the left margin, and displays each name within a package "qualified" by the name of the package that contains it. For example, if the name *A* is within package *P*, it is shown as *A.P.*, and if *A* in turn contains *X*, *X* is shown as *X.A.P.*)

Symbol-table considerations

Anyone making extensive use of packages should be aware of some practical issues that arise from the manner in which the APL system handles names. These aren't issues of language, but rather of logistics in the implementation — but they can make an important difference to how much it costs to run an application.

APL\360 adopted a design which has been retained in many (perhaps all?) of the systems descended from it, including SHARP APL. Each workspace contains a single symbol table. The symbol table contains one entry for each name in use anywhere in the workspace. That includes an entry for each name used to refer to a function or variable (or, while groups still exist, a group); it does **not** include the name of the workspace itself, or names used inside packages.

In the workspace, the internal representation of a function is called a **codestring.** Each name mentioned anywhere in the function is represented by a pointer to an entry in the symbol table. That's true for every name occurring anywhere in the function's definition (other than the text of a character constant or comment). It includes the names of functions that the definition may invoke, global variables, local variables, labels — and even misspellings that may be present in a defective definition.

The symbol table contains information which tells the system where in the workspace the visible referent of a name is to be found. When *X* is a variable, the symbol table shows where to find the value of *X*. But the symbol table has no way of recording function definitions that may use the name *X*. When you expunge the variable *X*, the system marks the entry *X* as having no visible referent. However, it cannot remove *X* from the symbol table because there may still be function definitions (or groups) that refer to *X*.

Each time you use a name that hasn't been used before, the symbol table gets a new entry. But when you expunge a name, that name nevertheless remains in the symbol table. If you keep on doing things that use names that have not been used before, eventually the symbol table is filled. Some of the names may no longer refer to anything. In APL\360, if you tried to use a new name when the symbol table had no space to put it, you got the message *SYMBOL TABLE FULL*. The only recourse was to *)SAVE* the workspace, *)CLEAR*, and *)COPY* the saved version (thereby building a new symbol table containing only names that were actually in use).

The Sharp system has automated that process under some conditions, but not all. The implementors (in their colourful way) call the process of purging no-longer-used names "symbol table garbage collect". During symbol table garbage collect, the system searches the entire workspace to identify names which are still present in the symbol table but are no longer referred to by anything. It frees those entries. Symbol table garbage collect is invoked each time you use the command *)SYMBOLS* to set the size of the symbol table. It is also invoked automatically if the system runs out of symbols while executing certain functions likely to create new names, such as ⍎ or *⎕FX* or *⎕PDEF* (but at the moment not *⎕SVO*). Since *⎕PDEF* may bring many new names into the workspace, *⎕PDEF* is especially likely to cause symbol table garbage collect. But since it is expensive to collect symbol table garbage, you need to understand when it may be required — and how it can be avoided. A badly designed application may require garbage collection every time it reads in a new package, although with only minor changes it might never need garbage collection at all.

Consider what happens when you use *⎕PDEF* to bring the contents of a package into the workspace. A package has its own symbol table, independent of the workspace around it. When you use *⎕PDEF*, the system first merges the package symbol table into the workspace symbol table. A name from the package which is already present in the workspace requires no change. A name that isn't yet there must be added to the symbol table. If there's no room for a new name, the system invokes symbol table garbage collect. That may free many entries, or none, depending on what objects have been expunged from the workspace. If after collecting the symbol table garbage there's still no room for the new name, the system reports the error *SYMBOL TABLE FULL*.

Once the workspace symbol table has been amended to include names used in the package, objects contained in the package are copied to the workspace. The codestring of each function transferred to the workspace is revised by substituting pointers to the workspace symbol table for pointers to the package symbol table; this substitution costs very little.

You can eliminate the need for symbol table garbage collection if your workspace has a symbol table large enough to contain **simultaneously** all the names that will ever be used in it. Even if the referents of those names won't all be present at the same time, you still need space for all the distinct names, since you want to avoid ever having to remove some names to make room for others.

There are several ingenious (if inelegant) tricks to make sure that the workspace has sufficient symbol table. One that I've used is to keep in the workspace a function which has no body (it is never used) but whose header contains every name from every package that will be *⎕PDEF*'d during use of the workspace. (That's not just the result of *⎕PNAMES* for each of those packages, but also all names referred to inside any of the functions included in the package.)

Packages vs. enclosed arrays

For a number of years there have been proposals to relax the restriction that each scalar in an array be a single number or character. If a scalar can have an interior structure which can be "disclosed" on demand to reveal an array inside, it becomes possible to form an array any or each of whose scalars is an enclosed array of any rank, shape, or type. There hasn't yet been complete agreement on how to do this, or even on what to call the proposed extensions; earlier publications used "general array" or "nested array", but more recently we seem to have settled on "enclosed array". Development is proceeding, and it seems likely that before too long many APL systems will permit an array to be made up of scalars formed by enclosing other arrays. Bob Bernecky and Ken Iverson have been preparing a paper on "Operators and Enclosed Arrays" for this conference; it appears elsewhere in this volume. However, these words are being written before I've had the benefit of reading it. With the prospect that enclosed arrays are close at hand, it becomes interesting to speculate how such a development may compare with packages. Perhaps packages will become obsolete.

Like a package, at least for the present, an enclosed array can't be directly displayed; a function something like the *DISP* function for displaying a package (described in an earlier section of this article) might be required. A default display for enclosed arrays may be developed; Bernecky and Iverson refer to some in their paper. On the other hand, when developed, the default display may provide ideas that will apply almost as well to packages.

I had been of the opinion that there's relatively little you can do with an enclosed array while it remains enclosed. (Bob Bernecky tells me that's because I haven't kept up with the work he and others have been doing, and invites me to the session at which he'll talk about enclosed arrays.) Current proposals include definitions for equality, membership, and index (dyadic iota) on enclosed arrays, but most other functions would apply to an enclosed array only if you first disclosed it. Much of the effort to develop techniques for utilizing enclosed arrays is aimed at defining operators that make it easier to apply an expression to the disclosed value of each scalar in an array (and then perhaps re-enclose the various results). Until we have such operators, the principal advantage of enclosing an array is that it permits you to keep that array together with other enclosed arrays — just as a package keeps disparate items together also.

The fundamental difference between a package and an array of enclosed scalars is **naming** vs. **indexing**. Arrays have structure; packages don't. The only way to select an object in a package is by its name. Within an array, a scalar (enclosed or not) has no name and is selected by its position. Extending APL to include enclosed arrays would not involve any of the novelties that reference by name has required in the handling of packages. Moreover, the existing structural rules that apply to simple arrays, (augmented with operators that apply functions systematically to the disclosed value of each scalar) will make it possible to apply an expression to each of the arrays within a nested array. The fact than an array has an explicit structure gives you a simple basis for applying an expression systematically across that structure. I find it hard to imagine an analogous extension that would apply an expression across the collection of names in a package.

Enclosed arrays (at least as far as I have been able to tell from current proposals) do not yet offer a means to include functions. Nor would they offer directly a means of creating a variety of objects simultaneously (which one now gets readily with *⎕PDEF*). Nor would enclosed arrays give the ability at the same time to expunge (which one gets by extracting from a package a name with no referent). Since all these are extremely useful in constructing overlays, it seems to me unlikely that enclosed arrays will replace packages as a means of writing program overlays, or of bringing in definitions for "shell" functions. But the existence of these convenient features of packages may exert a pressure on the language designers to provide some analogous capabilities with arrays, and especially to provide a way to enclose a function. Perhaps's they'll also permit a package to be enclosed.

It is my impression that packages were introduced at least in part as a sort of judicious compromise: they provided some of the benefits of enclosed arrays, but avoided the unresolved issues regarding them. By adopting selection-by-name, the designers ducked some unresolved issues in array theory and instead exploited an alternative already familiar in the workspace but not previously used to select part of a variable.

As operators are developed which permit expressions to be applied systematically to the various enclosed arrays within an array of scalars, I expect we'll see developments which will remain awkward or impossible to achieve with packages.

Conclusions

The introduction of packages has spectacularly simplified the writing of program overlays, and, more generally, of programs which bring in their own internal definitions from outside the workspace. These are accompanied by some irritating but not insurmountable problems with the workspace symbol table.

The fact that packages require reference by name rather than by position is a mixed blessing. There are situations where this seems appropriate, and others where it is awkward. Perhaps it calls further attention to some unresolved questions in APL about the scope of names.

With packages (and the package functions) it becomes possible to perform automatically most of the work formerly done with the commands)*GROUP* and)*COPY*. The direction of evolution is towards applications that require at most one workspace, and it seems likely that users will require far fewer private workspaces (but correspondingly greater storage in files).

Packages offer a sort of partial substitute for some of the benefits one can hope in future to receive from enclosed arrays. In particular, packages simplify the passing of disparate data in arguments, results, and shared variables. When APL systems offer enclosed arrays (and the operators needed to work with them easily) we'll have many capabilities that are not at present provided by packages. That's especially true of applications that require systematic treatment of all the arrays within in a single general array. On the other hand, packages provide some services about as well or better than can be expected of enclosed arrays (for example, passing of disparate arguments) and a few which enclosed arrays seem unlikely to duplicate (especially for function overlays).

Acknowledgments

I am grateful for information, comments, and suggestions received from David Allen, Bob Bernecky, Ken Chakahwata, Leigh Clayton, Bob Dabrowski, Doug Forkes, Leslie Goldsmith, Hugh Hyndman, Rohan Jayasekera, Roger Moore, Roland Pesch, Bill Royds, Mike Symes, and Paul Watters.

A FINANCIAL MODEL FOR ACQUISITION ANALYSIS

Don N. Hughes
Continental Telephone Service Corporation
Atlanta, Georgia

Introduction

Continental Telephone Corporation is the third largest independent telephone company in the United States. As is the case with all telephone companies, Continental is faced with the prospect of deregulation in the near future. Several years ago the management realized that the telephone industry was going to be deregulated and decided to address deregulation as an opportunity rather than as a threat. In order to capitalize on this opportunity they formed the Corporate Development Department of which I am a member. This department has the charter of formalizing the planning process and developing the strategies to capitalize on opportunities in the telecommunications marketplace. Part of this strategy is to grow through acquisitions. In order to determine the price to pay for an acquisition, and the contribution that the acquisition will make to Continental, we developed the Acquisition Financial Model. Through the use of this model we can project the acquisition's future earnings and cash streams, and compare these to past performance. These projections can then be compared to Continental's corporate objectives, thereby determining the acquisition's contribution to Continental's growth.

In developing this model we were fortunate that Corporate Development was a new and small department, thereby precluding time-consuming review cycles. The model set forth in this paper was developed in approximately 12 weeks by a second year MBA summer student, myself, and much appreciated support from the I.P. Sharp Atlanta office. It is interesting to note that prior to this project the summer student, who did the majority of the coding, had no APL experience.

Basic Requirements

The model needed to meet a number of requirements, which follow:

1. Project the traditional financial statements, including income statement, balance sheet, and source and use of funds.

2. Ability to put actuals and projections in the same model. By doing this, an analyst can readily compare projections and actuals. This would include not only financial accounts such as revenues, cost of sales, etc., but also financial ratios such as return on sales, days of receivables, etc.

3. Determine the net present value, and therefore, the recommended price of a potential acquisition using discounted cash flow (DCF) techniques.

4. Determine earnings per share (EPS), EPS dilution, and EPS contribution to the Continental long range plan. Provide the ability to do the EPS calculations, both dependent and independent of the financial statements. We wanted the ability to do EPS calculations beginning with "ball park" projections of earnings and funding needed and, later on, a more detailed analysis, supported by financial statements.

5. The initial information on a potential aquisition would come from secondary sources, usually an annual report and an SEC 10K. As in any financial analysis, the analyst normally begins with a limited amount of information and then, as the analysis proceeds, he obtains more and more detailed information. We wanted a model that would run on both limited and detailed information.

6. Perform "what if's" with a minimum of user inputs. Many models are built with little regard for this requirement and hence, when an analyst must do sensitivities, much of his time is spent making adjustments to the model so that the sensitivity analysis looks credible.

7. Ability to alter the time base with ease. Many times an analysis begins with a given time base only to find that management is interested in a different time base. Without the model being set up to change the time base with ease, it can be very time-consuming to alter the time base. In addition, we knew that annual reports only have two years of balance sheet data and once you make contact with a company and obtain historic balance sheets, you can get a better perspective of how well the company employs assets, manages it receivables and payables, etc. by reviewing the income statement and balance sheet ratios of past years.

8. Ability to modify the model with moderate ease. We knew this model would be continually modified as we became more familiar with the information generated and the needs of senior management.

System Design

One major reason why APL lends itself well to financial analysis is because it is a matrix oriented language which obviously fits the characteristics of financial statements, with the rows being accounts and the columns being the time base. Figure 1 shows the matrix assignment of the various financial functional areas in this model.

System Data Base Assignments

Matrix Assignment	Number of* Variables or Rows	Financial Functional Area
1	35	Income Statement
2	60	Balance Sheet
3	40	Source & Use of Funds
4	50	Forecast Factors
5	25	Ratio Analysis
6	25	Net Present Value — Price
7	60	Dilution

* Number of columns or years dependent on the needs of the analysis, workspace size is the limiting factor.

Figure 1

Each financial functional area is in its own matrix. By doing this, when a change is needed it is easy to determine which matrix needs changing and if one area is modified it will effect the other areas only to the extent that it interfaces the other functional areas. For instance, the income statement could be given more expense accounts and not effect the logic of the other functional areas. In another example, the method in which the net present value of a company is calculated can be, and in fact was, totally changed and had no impact on the logic and structure of the other functional areas including the ones to which it interfaces.

Each financial functional area has two sets of subroutines — one for historic calculations and another for projected calculations. Again, this isolates the various functional areas so that when one area is modified it will not effect the logic or structure of the other areas except where a direct interface with another functional area is involved.

A block diagram of the financial model is shown in Figure 2. Let's go through this block diagram. The historical data which initially comes from an annual report or 10K is input into the income and balance sheet matrices. From this data the model calculates source and use of funds which are calculated as year to year changes of the balance sheet. When the changes in the balance sheet do not equal the source and use of funds because funds flow between balance sheet time periods the additional funds flow show on the cash flow/deficit account on the source and use of funds statement (see Appendix A, part 5). From the historical data the model also calculates ratios for profitability, activity, leverage, and liquidity. These calculations include such ratios as the return on sales, days of receivables, debt to equity ratio, and the quick ratio. A complete list of these ratios is found in Appendix A, part 5, forecast factors.

The division between historic and projected data is done by user input. The user specifies which year is the first projected year. All years prior to this year are historic years and this year and all years after are projected years. From the model's perspective the difference between historic and projected data is the logic. Let's take taxes as an example. In historic years the calculation is taxes ÷ earnings before taxes = tax rate. In the projected years the calculation is earnings before taxes × tax rate = taxes. In the historic years the tax rate is a calculated variable and in the projected years it is an input. This feature has an unanticipated benefit when working with forecasts sent in by the potential acquisitions management. In this case the first projected year is set at the end of the forecast. The model calculates the ratio for the forecasted years. This data is saved and also transferred to a new case where adjustments are made to the ratios as deemed necessary by the analyst. The first projected year is then reset to the first actual projected year and the analyst has an internally generated forecast and a submitted forecast which can easily be compared. Most models are set up to use only factors as the inputs and therefore if actual data is submitted the user must go through the laborious process of converting this data to factor input just to get the data in the model. By having two sets of logic and a variable for the first projected year this unproductive step is eliminated.

After reviewing the historical ratios the analyst enters the appropriate ratios for the forecast time period (Figure 2, Section 4). Based on these financial ratios and projected revenue inputs (Figure 2, Section 1), both of which are input by the analyst, the model calculates the financial statements for the projected years. One may ask, "Why doesn't the model, through some sort of statistical averaging, project the financial ratios and have the analyst only provide the revenue forecast assumptions, at least for the initial run?" We considered this alternative but found it inappropriate for the types of businesses that Continental is pursuing. Many of these businesses are new and have little track record. In addition, an annual report may provide five years of historical income information, but only two years of balance sheet information. It was felt that reasonable projections for the future could only come from the rational judgment of an analyst, and not the mathematical projection of the past.

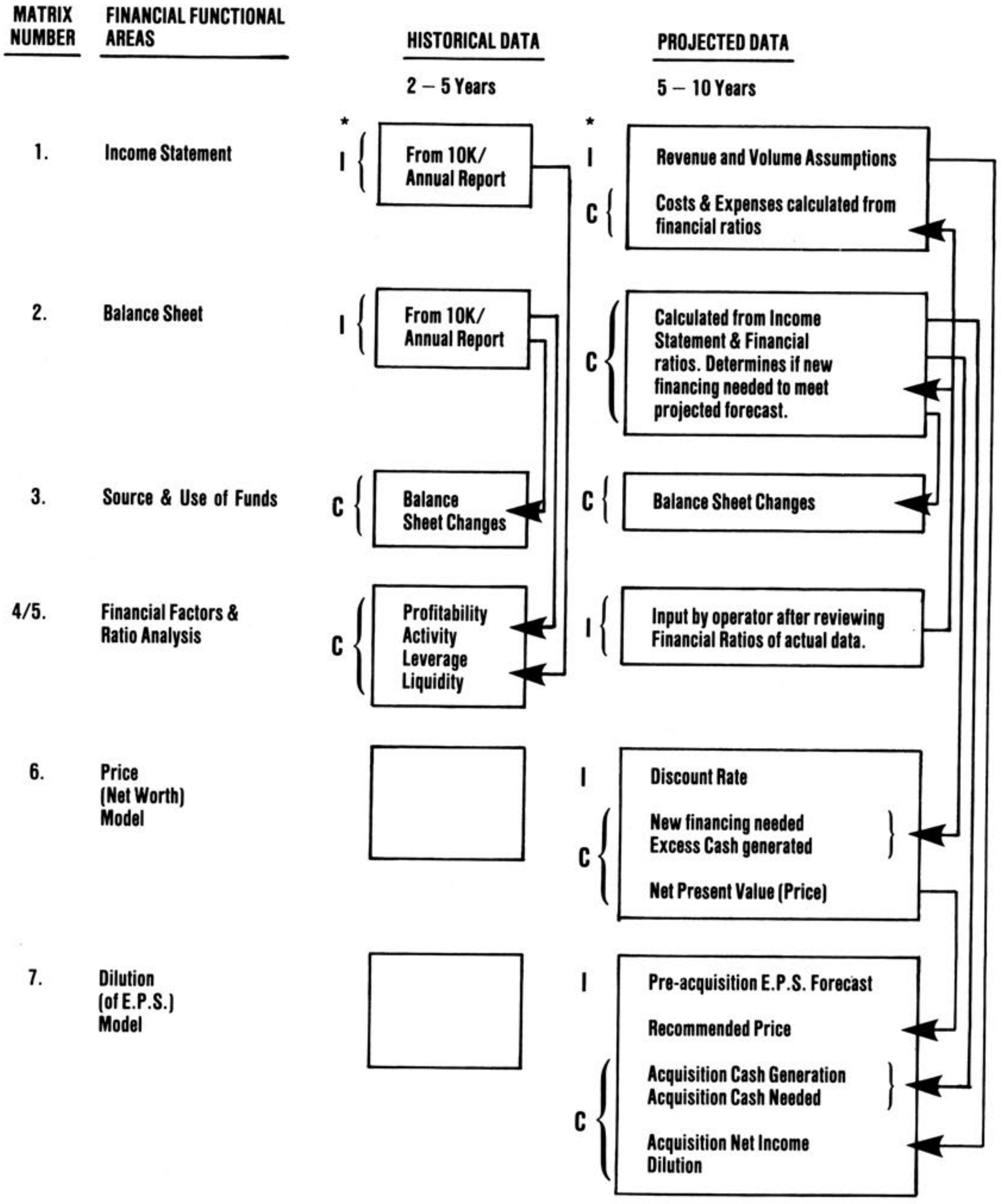
The output of the income statement is a net income stream which is fed to the dilution module (Figure 2, Section 7). The projected balance sheet (Figure 2, Section 2) is calculated from the income statement and financial ratios.

From the projected balance sheet the model determines the amount of new financing needed to sustain the projected revenue growth. This new financing is fed to the net present value (NPV) or pricing module and the dilution module (Figure 2, Sections 6 and 7). The NPV module using a discounted cash flow method determines the price to be paid which is fed to the dilution module. Using this (1) price from the NPV module, the (2) cash (needed) or generated from the balance sheet module and the (3) net income from the income statement module, the dilution module calculates the EPS dilution and contribution to Continental.

The only financial functional area remaining is the source and use of funds (Figure 2, Section 3). This statement is simply the year to year change of the balance sheet. Although it is the easiest to implement from a coding and mathematical standpoint, it is one of the most important statements for an analyst. From this statement it is easy to determine the amount of funds coming from income, new financing, and debt, and where these funds are being used, particularly for capital investment and the ratio analysis, an analyst can determine how well an acquisition employs its capital.

This system approach satisfies many of our basic requirements outlined in the previous section. The model projects traditional financial statements, accepts historic and projected data, and has a pricing and dilution module which may be operated independent of the rest of the system. By keeping the financial functional areas as independent from one another as possible, and using one data matrix per functional area the model can be changed with ease. To meet the requirement of altering the time base with ease, a user routine was developed. This routine allows the user to change the time base (lengthen or shorten), the only restriction being that at least one year of the old time base must be included in the new time base. In the case that the forecast portion of the time base is being increased, it allows the user to extend the data of the last year throughout the new projection years. This reduces the amount of data the user must input after making the time base adjustment.

ACQUISITION FINANCIAL MODEL
SYSTEM DESIGN OVERVIEW



MATRIX NUMBER	FINANCIAL FUNCTIONAL AREAS	HISTORICAL DATA 2 – 5 Years	PROJECTED DATA 5 – 10 Years
1.	Income Statement	* I From 10K/ Annual Report	* I Revenue and Volume Assumptions C Costs & Expenses calculated from financial ratios
2.	Balance Sheet	I From 10K/ Annual Report	C Calculated from Income Statement & Financial ratios. Determines if new financing needed to meet projected forecast.
3.	Source & Use of Funds	C Balance Sheet Changes	C Balance Sheet Changes
4/5.	Financial Factors & Ratio Analysis	C Profitability Activity Leverage Liquidity	I Input by operator after reviewing Financial Ratios of actual data.
6.	Price (Net Worth) Model		I Discount Rate C New financing needed Excess Cash generated Net Present Value (Price)
7.	Dilution (of E.P.S.) Model		I Pre-acquisition E.P.S. Forecast Recommended Price C Acquisition Cash Generation Acquisition Cash Needed Acquisition Net Income Dilution

* I = Operator Input
C = Calculated

Figure 2

Module Design

This section addresses the specifics of module design which the author feels would be of most interest to the reader. Each module is addressed in the sequence found in Figure 1.

One common mistake of many financial models is tying many of the expense accounts on the income statement to the revenue line. When management asks to see sensitivity to pricing and volume adjustments, the analyst spends much of his time juggling the inputs so that the fixed expenses do not vary with the revenue. In this model the revenue and cost and expense items are linked through two variables known as the number of variable units and the number of fixed units. Revenue is generated by multiplying the number of variable units times the retail price per unit. All variable expenses (indicated by a V after the title in Appendix A, part 1) and cost of sales are a function of variable units. To test the impact of a price change only the retail price variable is changed. By making revenue, variable expenses, and cost of goods sold a function of the variable units, one can simply increase the variable units by, let's say, 10% and see the impact of a more aggressive sales compaign. The second set of units known as fixed units is used to vary the fixed expenses. If the variable units increase dramatically, then at some point the fixed expenses must also be increased. This can be done by changing the number of fixed units at some lesser rate than the variable units. By changing these two input factors, the analyst will have a sensitivity to his base case supported with a balance sheet, source and use of funds, and EPS dilution calculations. These relationships can be seen by reviewing part 5, forecast factors, and part 1, the income statement of Appendix A.

The cost of goods sold is split between material and labor primarily so that the analyst can do sensitivities to the labour/material mix of cost of goods sold and have the model correctly impact the balance sheet and hence, working capital. The remaining income statement factors are straightforward and will be readily apparent to the reader.

Most of the balance sheet items are a function of the income statement. A quick glance at the forecast factors (Appendix A, part 5) will show that none of the balance sheet items are a function of revenue except accounts receivable. This again isolates pricing changes from impacting unrelated balance sheet items. Some of the factors such as days of inventory are a function of next year's cost of goods sold (next year is shown in the forecast factor schedule as t + 1). In many cases, especially businesses anticipating high growth, when assets are shown as a percentage of the current year's assets, working capital requirements are understated. When showing asset accounts, except for accounts receivable, as a function of next year's demand and the liability accounts as a function of this year's demand, working capital requirements are more accurately forecasted.

The quick ratio is used to determine the cash needed for working capital. The variable, cash as a % of cash + accounts receivable, keeps the cash account from becoming negative if accounts receivable by themselves are greater than the current liabilities. When accounts receivable are greater than current liabilities then a negative cash account would satisfy the quick ratio requirement if this variable is not used.

The most complex logic and one of the best features of the balance sheet module is the ability of the user to specify a target debt/equity ratio of the acquisition.

After calculating both sides of the balance sheet except financing needs, the model determines if the year being calculated is a cash generator or cash user. If it is a cash generator, over and above working capital needs, the cash is called excess cash on the balance sheet. On the other hand, if new financing is needed it must be split between debt and equity to achieve the debt/equity ratio specified by the user. The additional debt increases the interest payments and hence, the income statement is modified which flows through to retained earnings effecting the new financing needed on the balance sheet thereby making this an interative process.

In Appendix A, part 5, forecast factors, the requested debt to equity to maintain is 30%. Ratio analysis, Appendix A, part 7, shows that this ratio is maintained from 1980 through 1983 and thereafter the ratio is less than 30%. The source and use of funds, Appendix A, part 4, shows that in 1980 the acquisition will need a reduction in debt of $389,000 to maintain the 30% ratio while at the same time needing $915,000 in equity. This equity will come from the parent to its newly acquired subsidiary. By 1984, all financing can be done by the subsidiary itself through debt and no longer requires equity funds even though it has a requirement to pay off its outstanding debt at the rate of 12.5% per year. By 1986, it can meet both these requirements and begin generating excess cash as shown by the excess cash account in the balance sheet. By varying these key variables the user can study a number of financing and cashflow alternatives for this subsidiary. These variables include debt/equity ratio to maintain, common stock dividend payout ratio, and % payback of new debt.

At the bottom of the forecast factor schedule, there are a number of net add skew factors. These factors allow the user to weigh the value of an account over the course of the year. For example, debt may be taken on uniformly throughout the year so that a $1 million worth of new debt with an interest rate of 10% would have an interest cost of $50,000. In this case, the net add skew would be .5. If the $1 million in debt had been taken on January 1, the interest cost would have been $100,000 and the net add skew would have been 1. Likewise, if the debt was taken on December 31, the interest expense would be zero and the net add skew would be zero. The net add skew applies only to the new portion of the account taken on during the current year, and does not effect the account's beginning balance. Hence, in our previous example if there was $10 million dollars at the beginning of the year, by the end of the year there would be $11 million dollars of debt, irrespective of the net add skew.

The next module is the source and use of funds module which simply calculates the year to year change of the balance sheet items. The cash flow/deficit account will be zero unless the change between balance sheet accounts do not equal zero which is often the case in actual source and use of funds since funds flow between balance sheet time periods and hence, are not recorded on the balance sheet. Funds for both actual and projected years use the same logic.

The forecast factor module has separate logic for the actual and projected years. These forecast factors for the actuals years are calculated from the income statement and balance sheet and for the projected years they are user inputs. The ratio analysis module calculates standard financial ratios from the appropriate accounts. The debt to equity ratio and the quick ratio are found on both the forecast factors and ratio analysis schedules. These variables in the forecast factor schedule are requested ratios and in the ratio analysis schedule are the actual ratios. Therefore, many times they are different for reasons covered previously.

The net present value module is fed new financing needed which is a negative cash stream and excess cash which is a positive stream from the balance sheet. This is shown in Figure 2. The combination of these two is the net cash flow for the acquisition. This cash flow is then discounted to the present at a discount rate equal to or exceeding

Continental Telephone's cost of capital dependent on the acquisition's risk. The terminal value is assumed to be an annuity stream beginning in the last year of the forecast (the 10th year in Appendix A). The formula is shown in Figure 3.

$$\textbf{PRICE} = C_0 + \left(\sum_{t=1}^{n=T} \frac{C_t}{(1+k)^t} \right) + \frac{C_T}{(1+k)^T \times k} - D$$

C = Annual Cashflow
k = Discount Rate
D = Outstanding debt

Figure 3

The present value schedule is found in Appendix A, part 6. When the model is being used to calculate the net present value of the acquisition's cash stream, all financing variables must be set to zero since the net present value or recommended price is an investment decision and not a financing decision. Therefore the cash excess/(deficit) variable in the net present value schedule does not equal the sum of the new financing needed plus the excess cash of the balance sheet in Appendix A. If the debt/equity to maintain variable was set to zero these variables would be equal. The price deduction variable contains the outstanding debt in 1979 of the acquisition ($781,000 current + $384,000 long term = $1,165,000 total). The recommended price is equal to the excess cash + the salvage value - the price deduction. The recommended price for the example in Appendix A is $6,923,000. This price is fed to the dilution module.

The dilution schedule (Appendix A, part 8) shows the acquisition's dilution and, hopefully, eventual contribution to the acquiror's income stream. The acquiror's income stream and shares outstanding determine the pre-acquisition earnings per share stream. The dilution module is fed acquisition income from the income statement module and cash flow (titled "new fin (sub equity)" on the dilution schedule) from the balance sheet module. This cash flow is the sum of new financing needed and excess cash. This new financing is that needed above the amount that the acquisition could obtain through its own use of debt and maintain a 30% debt/equity ratio.

It is important to remember that the net present value module determines the worth of an investment, while the dilution module considers various methods of financing that investment.

Assumptions are made by the analyst for stock prices, dividend requirements, the method of purchase price payment, the debt to equity split for the new financing, and the acquiror's cost of debt. Our example assumes a pooling of interest and hence, 100% equity for the price. If it were a purchased acquisition, then the cost in excess of book value would have to be amortized against income after the assets had been revalued.

The funding of the new financing required is split 60% common stock, 10% preferred stock, and 30% debt. As you would expect, the new financing of the subsidiary on the dilution schedule equals the change in total stock/equity on the source and use of funds statement of the acquisition. Additional new financing is needed to support the acquiror's dividend policy. The sum of these two financing requirements is the total new financing needed which is funded by the 60/10/30 split mentioned above.

The dilution per share-cents variable shows the acquisition's dilution, and then contribution to the acquiror's earnings per share stream. The acquisition EPS variable is the earnings of this acquisition divided by the number of outstanding shares issued for its purchase and funding. This variable shows how well the acquisition contributes to earnings compared to the pre-acquisition earnings per share. The acquisition will be dilutive until its EPS equals or exceeds the preacquisition EPS of the acquiror. This variable helps you see the contribution to earnings where the dilution per share can be clouded because of the relative magnitude between the pre-acquisition income of the acquiror and the income of the acquisition. The acquiror's and acquisition's contribution to total EPS under the heading EPS contribution is another way of showing the relative weighting of the acquisition's contribution to the acquiror's total earnings.

The dilution schedule points out a number of important factors which are often overlooked in acquisition analysis. Many times only earnings and the common stock issued for the purchase are part of the analysis. The acquisition's earnings are only one factor. The funding required to support the acquisition's growth and its eventual cash contributions are just as important as earnings. Probably the reason these items are often overlooked is because the funding requirements are difficult to estimate. Only through a model, such as the one outlined in this article, can these requirements be determined in a reasonable period of time.

In our example, the earnings contribution in the early years (1980 - 1983) is partially offset by the funding required and the earnings in the later years (1986 - 1990) is enhanced by the cash contribution of the acquisition. By 1988 the acquisition's excess cash flow is paying off the debt used to finance it in the early years and is buying back the stock issued for its purchase and funding. This excess cash flow contributes to increased EPS. As you can see from the dilution schedule there are many alternatives to financing this acquisition. Many are less dilutive than others — but no matter what the financing method, the value of the acquisition and hence, the price the acquiror should be willing to pay remains unchanged.

This module can be run independent of the other modules and has proven to be quite useful in reviewing various purchase prices, earnings streams, and funding levels. As you move through acquisition negotiations, this schedule allows you to see the impact of various pricing strategies before they are proposed to the acquisition candidate's management.

Summary

This acquisition analysis model was developed to assist the management at Continental Telephone Corporation in valuing potential acquisitions and showing the acquisition's contribution to Continental's growth. It was developed in approximately twelve weeks by a second year MBA summer student, myself, and support from the I.P. Sharp Atlanta office.

The majority of the coding was done by the summer student who had no prior APL experience. The only experience he had with programming was a couple of timesharing classes which were part of his MBA course work. If APL financial models are designed and managed by an experienced APL user, the knowledge of APL of the person coding the majority of the logic does not need to be more than simple matrix manipulation.

The more complex routines, such as DDB depreciation, loan amortizations, etc., will have to be developed by an experienced programmer if they are not already available in the timesharing library.

This model's modular design was easy to code and is easy to maintain. This is important because models are continually modified as the user becomes more familiar with their capabilities.

By segmenting the financial functional areas, certain modules, such as the dilution module, can be run independent of the rest of the system. This enables the analyst to run "ball park" numbers initially and then become more precise through various phases of the analysis.

Throughout the design of this model, we strived to make it as user-oriented as possible. The variables are set up so that only a minimum number need changing when running sensitivities. The model will run on ten input line items, yet has the ability to accept over a 100 line items for more detailed analysis. Both historic and projected data is accepted in the same data base. This saves the user from building and maintaining two data bases which must be merged for printouts. Since the historic data logic accepts the actual account data such as taxes, and the projected data logic accepts factors, such as the tax rate, either type of forecast submitted to the user is accepted by the model. The most important feature of a model is its user interface. For if a model is not accepted by the user then all the effort in developing the model is wasted as it gathers dust on the shelf of a tape locker.

Appendix A — Part 1

FILE : DEMO
REPORT 1
CASE 7   TORONTO
77 -   88

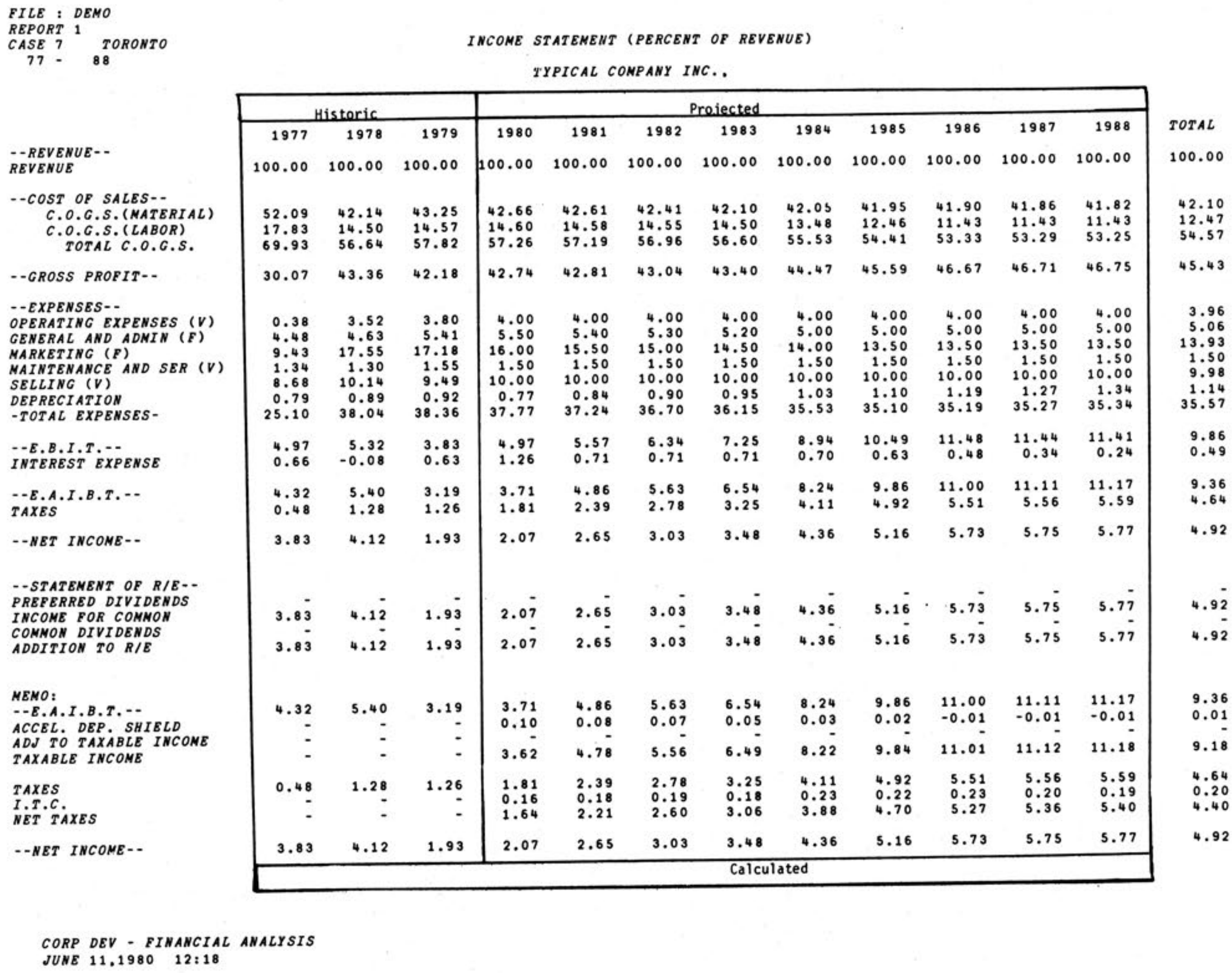
INCOME STATEMENT

TYPICAL COMPANY INC.,

	Historic			Projected									
	1977	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	TOTAL
--REVENUE--													
REVENUE	3249	4724	7875	11518	15005	19667	25976	34177	44837	57903	72467	87745	385142
--COST OF SALES--													
C.O.G.S.(MATERIAL)	1692	1991	3405	4914	6394	8341	10936	14371	18809	24261	30335	36695	162144
C.O.G.S.(LABOR)	579	685	1147	1682	2188	2862	3767	4607	5587	6618	8283	10029	48033
TOTAL C.O.G.S.	2272	2676	4553	6595	8581	11202	14702	18978	24396	30880	38618	46724	210177
--GROSS PROFIT--	977	2048	3322	4923	6424	8465	11274	15199	20441	27023	33849	41021	174965
--EXPENSES--													
OPERATING EXPENSES (V)	12	166	299	461	600	787	1039	1367	1793	2316	2899	3510	15250
GENERAL AND ADMIN (F)	146	219	426	633	810	1042	1351	1709	2242	2895	3623	4387	19484
MARKETING (F)	306	829	1352	1843	2326	2950	3767	4785	6053	7817	9783	11846	53657
MAINTENANCE AND SER (V)	43	61	122	173	225	295	390	513	673	869	1087	1316	5767
SELLING (V)	282	479	747	1152	1501	1967	2598	3418	4484	5790	7247	8775	38438
DEPRECIATION	26	42	73	88	126	178	247	350	495	687	917	1176	4405
-TOTAL EXPENSES-	815	1797	3020	4350	5588	7219	9391	12141	15740	20374	25556	31009	136999
--E.B.I.T.--	162	251	301	573	836	1246	1883	3057	4701	6649	8293	10012	37965
INTEREST EXPENSE	21	-4	50	146	106	139	184	239	282	277	245	215	1901
--E.A.I.B.T.--	140	255	252	427	730	1107	1699	2818	4420	6372	8048	9797	36065
TAXES	16	60	99	208	359	547	843	1404	2206	3188	4028	4904	17862
--NET INCOME--	125	195	152	238	398	597	903	1491	2312	3319	4165	5062	18956
--STATEMENT OF R/E--													
PREFERRED DIVIDENDS	-	-	-	-	-	-	-	-	-	-	-	-	-
INCOME FOR COMMON	125	195	152	238	398	597	903	1491	2312	3319	4165	5062	18956
COMMON DIVIDENDS	-	-	-	-	-	-	-	-	-	-	-	-	-
ADDITION TO R/E	125	195	152	238	398	597	903	1491	2312	3319	4165	5062	18956
MEMO:													
--E.A.I.B.T.--	140	255	252	427	730	1107	1699	2818	4420	6372	8048	9797	36065
ACCEL. DEP. SHIELD	-	-	-	11	13	13	12	9	7	-3	-7	-11	44
ADJ TO TAXABLE INCOME	-	-	-	-	-	-	-	-	-	-	-	-	-
TAXABLE INCOME	-	-	-	416	717	1094	1687	2809	4412	6375	8055	9808	35374
TAXES	16	60	99	208	359	547	843	1404	2206	3188	4028	4904	17862
I.T.C.	-	-	-	19	27	36	48	77	98	134	144	169	753
NET TAXES	-	-	-	189	332	511	796	1327	2108	3053	3883	4735	16934
--NET INCOME--	125	195	152	238	398	597	903	1491	2312	3319	4165	5062	18956
	Input			Calculated									

CORP DEV - FINANCIAL ANALYSIS
JUNE 11,1980 10:53

Appendix A — Part 2



FILE : DEMO
REPORT 1
CASE 7 TORONTO
77 - 88

INCOME STATEMENT (PERCENT OF REVENUE)

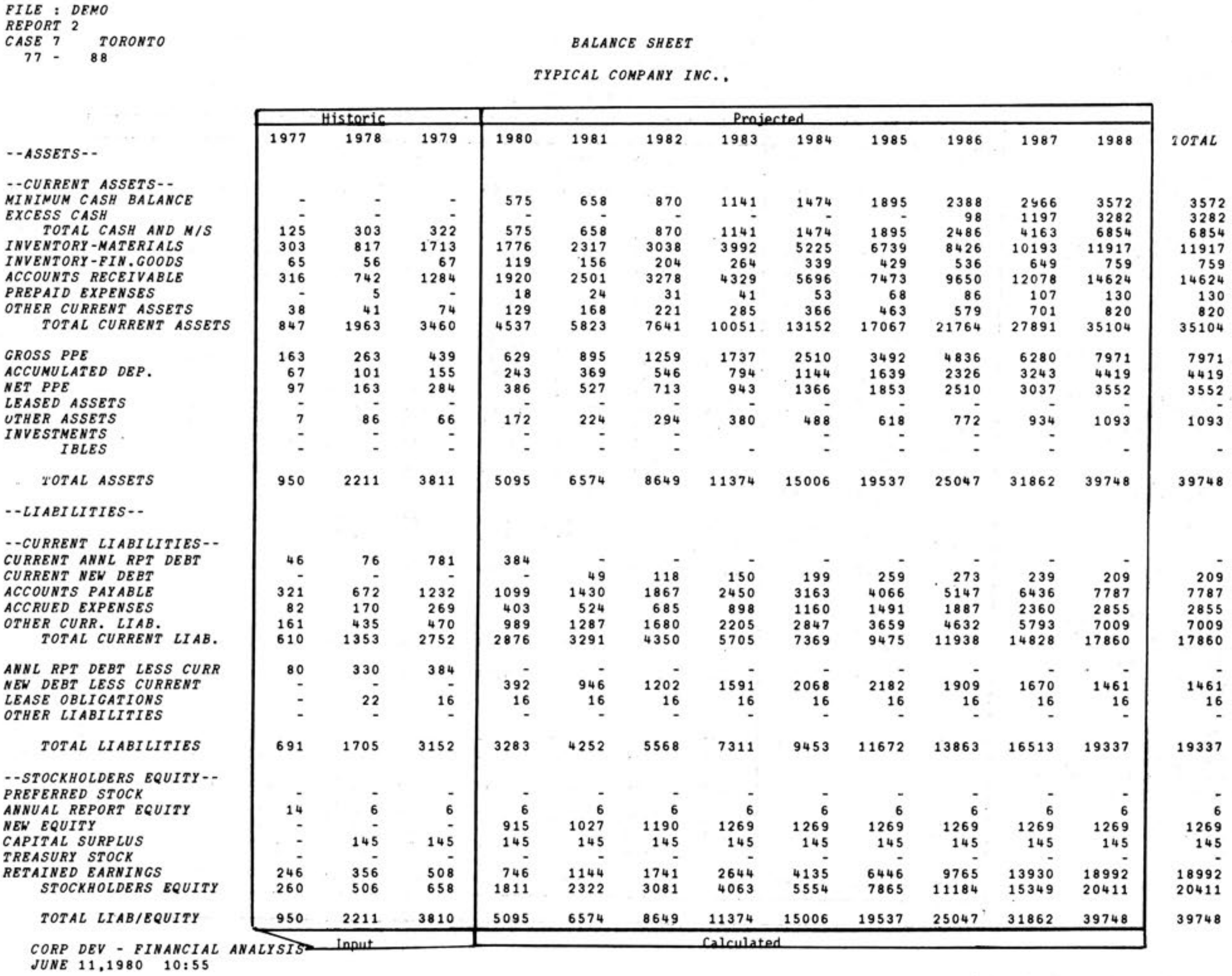
TYPICAL COMPANY INC..

	Historic			Projected									
	1977	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	TOTAL
--REVENUE--													
REVENUE	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
--COST OF SALES--													
C.O.G.S.(MATERIAL)	52.09	42.14	43.25	42.66	42.61	42.41	42.10	42.05	41.95	41.90	41.86	41.82	42.10
C.O.G.S.(LABOR)	17.83	14.50	14.57	14.60	14.58	14.55	14.50	13.48	12.46	11.43	11.43	11.43	12.47
TOTAL C.O.G.S.	69.93	56.64	57.82	57.26	57.19	56.96	56.60	55.53	54.41	53.33	53.29	53.25	54.57
--GROSS PROFIT--	30.07	43.36	42.18	42.74	42.81	43.04	43.40	44.47	45.59	46.67	46.71	46.75	45.43
--EXPENSES--													
OPERATING EXPENSES (V)	0.38	3.52	3.80	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	3.96
GENERAL AND ADMIN (F)	4.48	4.63	5.41	5.50	5.40	5.30	5.20	5.00	5.00	5.00	5.00	5.00	5.06
MARKETING (F)	9.43	17.55	17.18	16.00	15.50	15.00	14.50	14.00	13.50	13.50	13.50	13.50	13.93
MAINTENANCE AND SER (V)	1.34	1.30	1.55	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50
SELLING (V)	8.68	10.14	9.49	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	9.98
DEPRECIATION	0.79	0.89	0.92	0.77	0.84	0.90	0.95	1.03	1.10	1.19	1.27	1.34	1.14
-TOTAL EXPENSES-	25.10	38.04	38.36	37.77	37.24	36.70	36.15	35.53	35.10	35.19	35.27	35.34	35.57
--E.B.I.T.--	4.97	5.32	3.83	4.97	5.57	6.34	7.25	8.94	10.49	11.48	11.44	11.41	9.86
INTEREST EXPENSE	0.66	-0.08	0.63	1.26	0.71	0.71	0.71	0.70	0.63	0.48	0.34	0.24	0.49
--E.A.I.B.T.--	4.32	5.40	3.19	3.71	4.86	5.63	6.54	8.24	9.86	11.00	11.11	11.17	9.36
TAXES	0.48	1.28	1.26	1.81	2.39	2.78	3.25	4.11	4.92	5.51	5.56	5.59	4.64
--NET INCOME--	3.83	4.12	1.93	2.07	2.65	3.03	3.48	4.36	5.16	5.73	5.75	5.77	4.92
--STATEMENT OF R/E--													
PREFERRED DIVIDENDS	-	-	-	-	-	-	-	-	-	-	-	-	-
INCOME FOR COMMON	3.83	4.12	1.93	2.07	2.65	3.03	3.48	4.36	5.16	5.73	5.75	5.77	4.92
COMMON DIVIDENDS	-	-	-	-	-	-	-	-	-	-	-	-	-
ADDITION TO R/E	3.83	4.12	1.93	2.07	2.65	3.03	3.48	4.36	5.16	5.73	5.75	5.77	4.92
MEMO:													
--E.A.I.B.T.--	4.32	5.40	3.19	3.71	4.86	5.63	6.54	8.24	9.86	11.00	11.11	11.17	9.36
ACCEL. DEP. SHIELD	-	-	-	0.10	0.08	0.07	0.05	0.03	0.02	-0.01	-0.01	-0.01	0.01
ADJ TO TAXABLE INCOME	-	-	-	-	-	-	-	-	-	-	-	-	-
TAXABLE INCOME	-	-	-	3.62	4.78	5.56	6.49	8.22	9.84	11.01	11.12	11.18	9.18
TAXES	0.48	1.28	1.26	1.81	2.39	2.78	3.25	4.11	4.92	5.51	5.56	5.59	4.64
I.T.C.	-	-	-	0.16	0.18	0.19	0.18	0.23	0.22	0.23	0.20	0.19	0.20
NET TAXES	-	-	-	1.64	2.21	2.60	3.06	3.88	4.70	5.27	5.36	5.40	4.40
--NET INCOME--	3.83	4.12	1.93	2.07	2.65	3.03	3.48	4.36	5.16	5.73	5.75	5.77	4.92

Calculated

CORP DEV - FINANCIAL ANALYSIS
JUNE 11,1980 12:18

Appendix A — Part 3



FILE : DEMO
REPORT 2
CASE 7   TORONTO
77 -   88

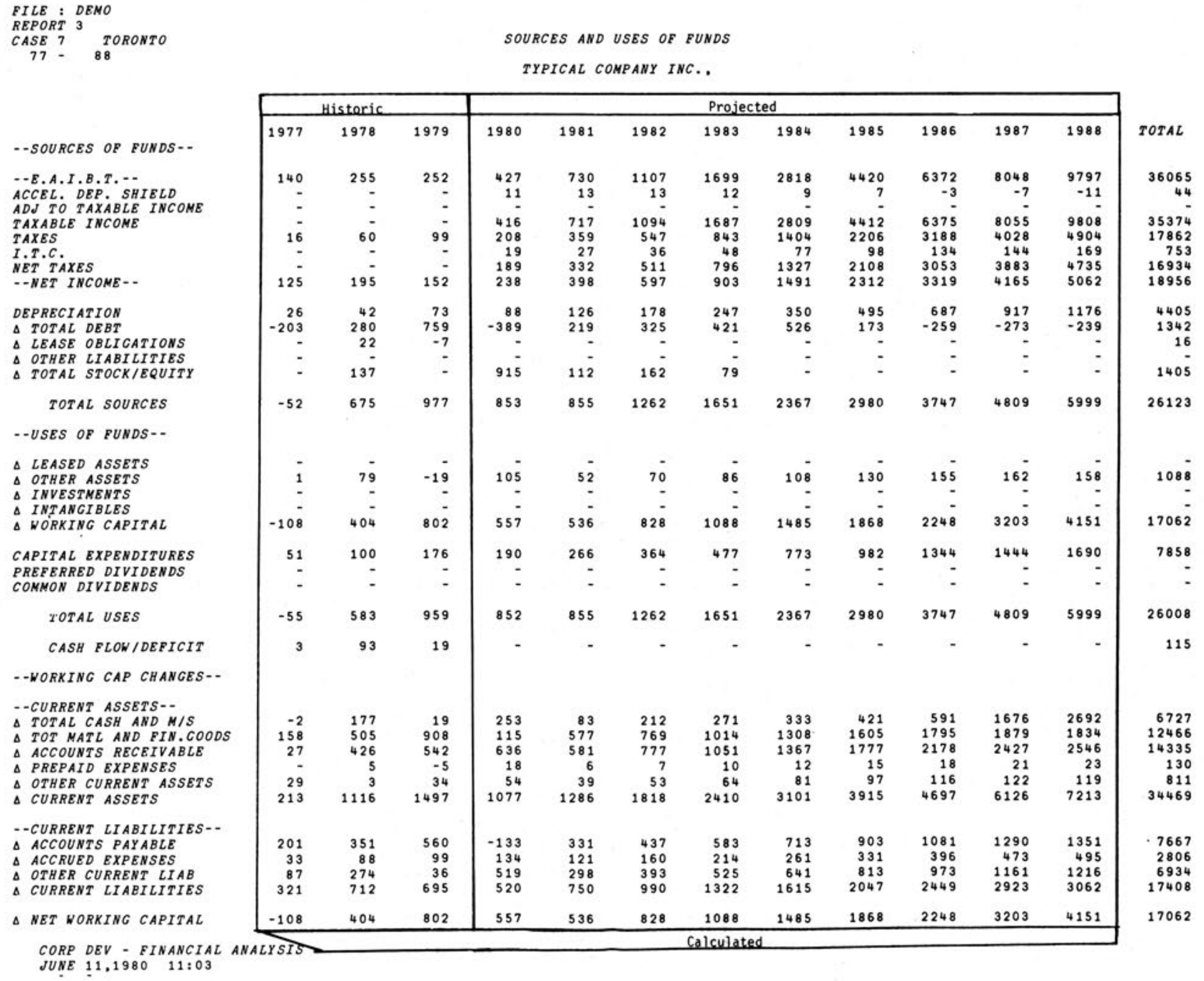
BALANCE SHEET

TYPICAL COMPANY INC..

	Historic			Projected									
	1977	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	TOTAL
--ASSETS--													
--CURRENT ASSETS--													
MINIMUM CASH BALANCE	-	-	-	575	658	870	1141	1474	1895	2388	2966	3572	3572
EXCESS CASH	-	-	-	-	-	-	-	-	-	98	1197	3282	3282
TOTAL CASH AND M/S	125	303	322	575	658	870	1141	1474	1895	2486	4163	6854	6854
INVENTORY-MATERIALS	303	817	1713	1776	2317	3038	3992	5225	6739	8426	10193	11917	11917
INVENTORY-FIN.GOODS	65	56	67	119	156	204	264	339	429	536	649	759	759
ACCOUNTS RECEIVABLE	316	742	1284	1920	2501	3278	4329	5696	7473	9650	12078	14624	14624
PREPAID EXPENSES	-	5	-	18	24	31	41	53	68	86	107	130	130
OTHER CURRENT ASSETS	38	41	74	129	168	221	285	366	463	579	701	820	820
TOTAL CURRENT ASSETS	847	1963	3460	4537	5823	7641	10051	13152	17067	21764	27891	35104	35104
GROSS PPE	163	263	439	629	895	1259	1737	2510	3492	4836	6280	7971	7971
ACCUMULATED DEP.	67	101	155	243	369	546	794	1144	1639	2326	3243	4419	4419
NET PPE	97	163	284	386	527	713	943	1366	1853	2510	3037	3552	3552
LEASED ASSETS	-	-	-	-	-	-	-	-	-	-	-	-	-
UTHER ASSETS	7	86	66	172	224	294	380	488	618	772	934	1093	1093
INVESTMENTS	-	-	-	-	-	-		-	-	-	-		
IBLES	-	-	-	-	-	-	-	-	-	-	-	-	-
TOTAL ASSETS	950	2211	3811	5095	6574	8649	11374	15006	19537	25047	31862	39748	39748
--LIABILITIES--													
--CURRENT LIABILITIES--													
CURRENT ANNL RPT DEBT	46	76	781	384	-	-	-	-	-	-	-	-	-
CURRENT NEW DEBT	-	-	-	-	49	118	150	199	259	273	239	209	209
ACCOUNTS PAYABLE	321	672	1232	1099	1430	1867	2450	3163	4066	5147	6436	7787	7787
ACCRUED EXPENSES	82	170	269	403	524	685	898	1160	1491	1887	2360	2855	2855
OTHER CURR. LIAB.	161	435	470	989	1287	1680	2205	2847	3659	4632	5793	7009	7009
TOTAL CURRENT LIAB.	610	1353	2752	2876	3291	4350	5705	7369	9475	11938	14828	17860	17860
ANNL RPT DEBT LESS CURR	80	330	384	-	-	-	-	-	-	-	-	-	-
NEW DEBT LESS CURRENT	-	-	-	392	946	1202	1591	2068	2182	1909	1670	1461	1461
LEASE OBLIGATIONS	-	22	16	16	16	16	16	16	16	16	16	16	16
OTHER LIABILITIES	-	-	-	-	-	-	-	-	-	-	-	-	-
TOTAL LIABILITIES	691	1705	3152	3283	4252	5568	7311	9453	11672	13863	16513	19337	19337
--STOCKHOLDERS EQUITY--													
PREFERRED STOCK	-	-	-	-	-	-	-	-	-	-	-	-	-
ANNUAL REPORT EQUITY	14	6	6	6	6	6	6	6	6	6	6	6	6
NEW EQUITY	-	-	-	915	1027	1190	1269	1269	1269	1269	1269	1269	1269
CAPITAL SURPLUS	-	145	145	145	145	145	145	145	145	145	145	145	145
TREASURY STOCK	-	-	-	-	-	-	-	-	-	-	-	-	-
RETAINED EARNINGS	246	356	508	746	1144	1741	2644	4135	6446	9765	13930	18992	18992
STOCKHOLDERS EQUITY	260	506	658	1811	2322	3081	4063	5554	7865	11184	15349	20411	20411
TOTAL LIAB/EQUITY	950	2211	3810	5095	6574	8649	11374	15006	19537	25047	31862	39748	39748
	Input			Calculated									

CORP DEV - FINANCIAL ANALYSIS
JUNE 11,1980 10:55

Appendix A — Part 4



FILE : DEMO
REPORT 3
CASE 7 TORONTO
77 - 88

SOURCES AND USES OF FUNDS

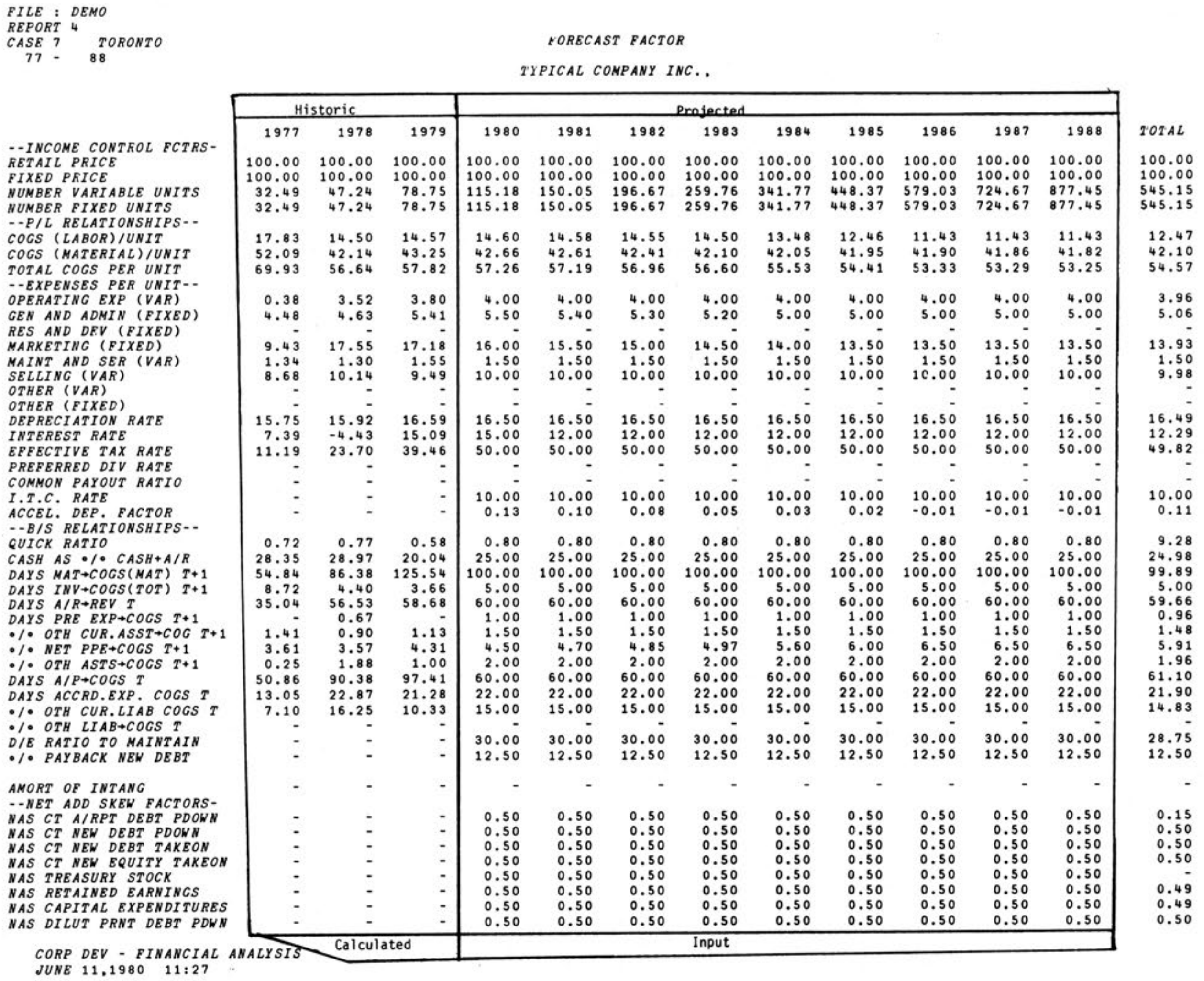
TYPICAL COMPANY INC.,

	Historic			Projected									
	1977	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	TOTAL
--SOURCES OF FUNDS--													
--E.A.I.B.T.--	140	255	252	427	730	1107	1699	2818	4420	6372	8048	9797	36065
ACCEL. DEP. SHIELD	-	-	-	11	13	13	12	9	7	-3	-7	-11	44
ADJ TO TAXABLE INCOME	-	-	-	-	-	-	-	-	-	-	-	-	-
TAXABLE INCOME	-	-	-	416	717	1094	1687	2809	4412	6375	8055	9808	35374
TAXES	16	60	99	208	359	547	843	1404	2206	3188	4028	4904	17862
I.T.C.	-	-	-	19	27	36	48	77	98	134	144	169	753
NET TAXES	-	-	-	189	332	511	796	1327	2108	3053	3883	4735	16934
--NET INCOME--	125	195	152	238	398	597	903	1491	2312	3319	4165	5062	18956
DEPRECIATION	26	42	73	88	126	178	247	350	495	687	917	1176	4405
Δ TOTAL DEBT	-203	280	759	-389	219	325	421	526	173	-259	-273	-239	1342
Δ LEASE OBLIGATIONS	-	22	-7	-	-	-	-	-	-	-	-	-	16
Δ OTHER LIABILITIES	-	-	-	-	-	-	-	-	-	-	-	-	-
Δ TOTAL STOCK/EQUITY	-	137	-	915	112	162	79	-	-	-	-	-	1405
TOTAL SOURCES	-52	675	977	853	855	1262	1651	2367	2980	3747	4809	5999	26123
--USES OF FUNDS--													
Δ LEASED ASSETS	-	-	-	-	-	-	-	-	-	-	-	-	-
Δ OTHER ASSETS	1	79	-19	105	52	70	86	108	130	155	162	158	1088
Δ INVESTMENTS	-	-	-	-	-	-	-	-	-	-	-	-	-
Δ INTANGIBLES	-	-	-	-	-	-	-	-	-	-	-	-	-
Δ WORKING CAPITAL	-108	404	802	557	536	828	1088	1485	1868	2248	3203	4151	17062
CAPITAL EXPENDITURES	51	100	176	190	266	364	477	773	982	1344	1444	1690	7858
PREFERRED DIVIDENDS	-	-	-	-	-	-	-	-	-	-	-	-	-
COMMON DIVIDENDS	-	-	-	-	-	-	-	-	-	-	-	-	-
TOTAL USES	-55	583	959	852	855	1262	1651	2367	2980	3747	4809	5999	26008
CASH FLOW/DEFICIT	3	93	19	-	-	-	-	-	-	-	-	-	115
--WORKING CAP CHANGES--													
--CURRENT ASSETS--													
Δ TOTAL CASH AND M/S	-2	177	19	253	83	212	271	333	421	591	1676	2692	6727
Δ TOT MATL AND FIN.COODS	158	505	908	115	577	769	1014	1308	1605	1795	1879	1834	12466
Δ ACCOUNTS RECEIVABLE	27	426	542	636	581	777	1051	1367	1777	2178	2427	2546	14335
Δ PREPAID EXPENSES	-	5	-5	18	6	7	10	12	15	18	21	23	130
Δ OTHER CURRENT ASSETS	29	3	34	54	39	53	64	81	97	116	122	119	811
Δ CURRENT ASSETS	213	1116	1497	1077	1286	1818	2410	3101	3915	4697	6126	7213	34469
--CURRENT LIABILITIES--													
Δ ACCOUNTS PAYABLE	201	351	560	-133	331	437	583	713	903	1081	1290	1351	7667
Δ ACCRUED EXPENSES	33	88	99	134	121	160	214	261	331	396	473	495	2806
Δ OTHER CURRENT LIAB	87	274	36	519	298	393	525	641	813	973	1161	1216	6934
Δ CURRENT LIABILITIES	321	712	695	520	750	990	1322	1615	2047	2449	2923	3062	17408
Δ NET WORKING CAPITAL	-108	404	802	557	536	828	1088	1485	1868	2248	3203	4151	17062

Calculated

CORP DEV - FINANCIAL ANALYSIS
JUNE 11,1980 11:03

Appendix A — Part 5



FILE : DEMO
REPORT 4
CASE 7 TORONTO
77 - 88

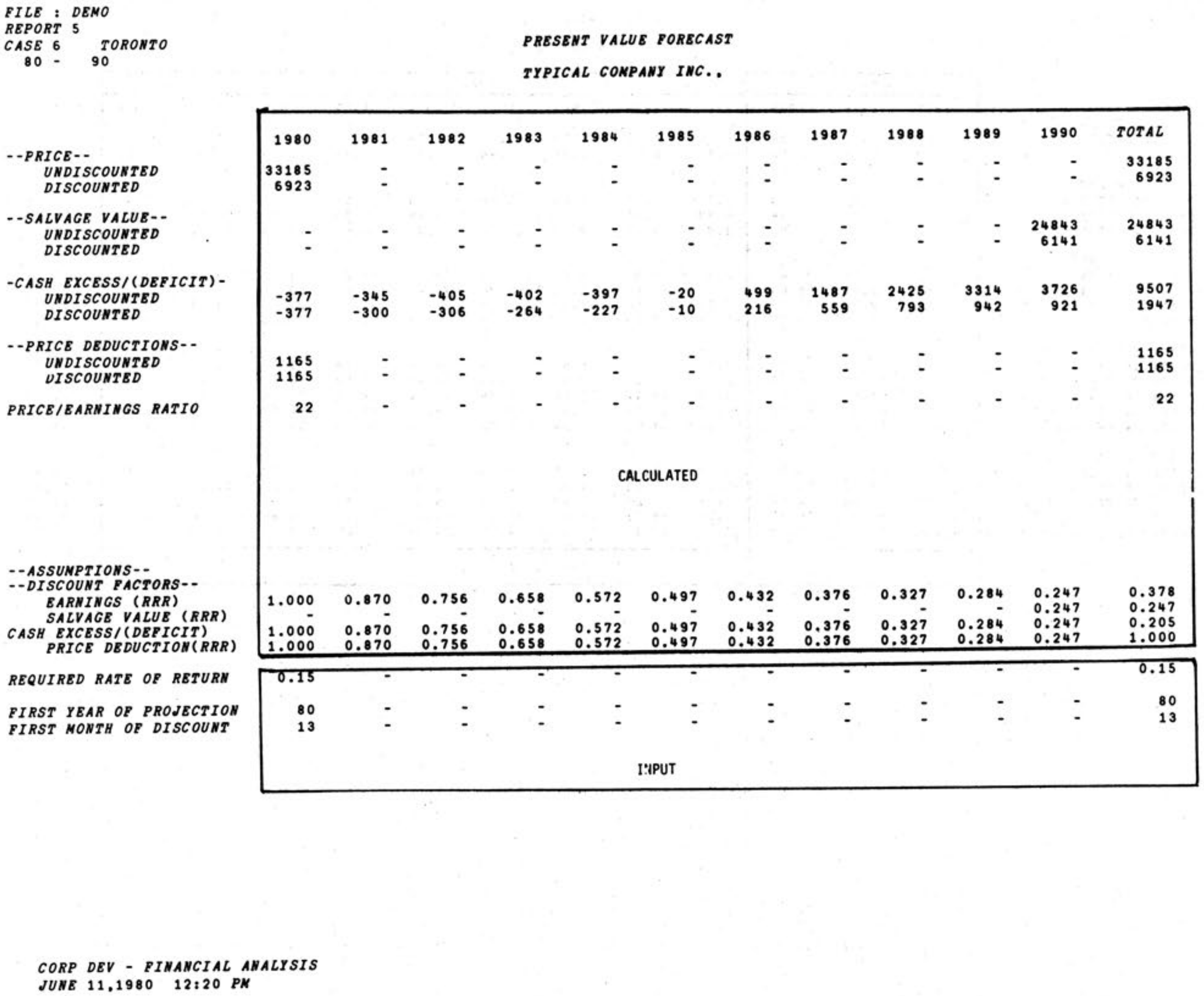
FORECAST FACTOR

TYPICAL COMPANY INC.,

	Historic			Projected									
	1977	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	TOTAL
--INCOME CONTROL FCTRS-													
RETAIL PRICE	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
FIXED PRICE	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
NUMBER VARIABLE UNITS	32.49	47.24	78.75	115.18	150.05	196.67	259.76	341.77	448.37	579.03	724.67	877.45	545.15
NUMBER FIXED UNITS	32.49	47.24	78.75	115.18	150.05	196.67	259.76	341.77	448.37	579.03	724.67	877.45	545.15
--P/L RELATIONSHIPS--													
COGS (LABOR)/UNIT	17.83	14.50	14.57	14.60	14.58	14.55	14.50	13.48	12.46	11.43	11.43	11.43	12.47
COGS (MATERIAL)/UNIT	52.09	42.14	43.25	42.66	42.61	42.41	42.10	42.05	41.95	41.90	41.86	41.82	42.10
TOTAL COGS PER UNIT	69.93	56.64	57.82	57.26	57.19	56.96	56.60	55.53	54.41	53.33	53.29	53.25	54.57
--EXPENSES PER UNIT--													
OPERATING EXP (VAR)	0.38	3.52	3.80	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	3.96
GEN AND ADMIN (FIXED)	4.48	4.63	5.41	5.50	5.40	5.30	5.20	5.00	5.00	5.00	5.00	5.00	5.06
RES AND DEV (FIXED)	-	-	-	-	-	-	-	-	-	-	-	-	-
MARKETING (FIXED)	9.43	17.55	17.18	16.00	15.50	15.00	14.50	14.00	13.50	13.50	13.50	13.50	13.93
MAINT AND SER (VAR)	1.34	1.30	1.55	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50
SELLING (VAR)	8.68	10.14	9.49	10.00	10.00	10.00	10.00	10.00	10.00	1C.00	10.00	10.00	9.98
OTHER (VAR)	-	-	-	-	-	-	-	-	-	-	-	-	-
OTHER (FIXED)	-	-	-	-	-	-	-	-	-	-	-	-	-
DEPRECIATION RATE	15.75	15.92	16.59	16.50	16.50	16.50	16.50	16.50	16.50	16.50	16.50	16.50	16.49
INTEREST RATE	7.39	-4.43	15.09	15.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.29
EFFECTIVE TAX RATE	11.19	23.70	39.46	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	49.82
PREFERRED DIV RATE	-	-	-	-	-	-	-	-	-	-	-	-	-
COMMON PAYOUT RATIO	-	-	-	-	-	-	-	-	-	-	-	-	-
I.T.C. RATE	-	-	-	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00
ACCEL. DEP. FACTOR	-	-	-	0.13	0.10	0.08	0.05	0.03	0.02	-0.01	-0.01	-0.01	0.11
--B/S RELATIONSHIPS--													
QUICK RATIO	0.72	0.77	0.58	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	9.28
CASH AS •/• CASH+A/R	28.35	28.97	20.04	25.00	25.00	25.00	25.00	25.00	25.00	25.00	25.00	25.00	24.98
DAYS MAT→COGS(MAT) T+1	54.84	86.38	125.54	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	99.89
DAYS INV→COGS(TOT) T+1	8.72	4.40	3.66	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00
DAYS A/R→REV T	35.04	56.53	58.68	60.00	60.00	60.00	60.00	60.00	60.00	60.00	60.00	60.00	59.66
DAYS PRE EXP→COGS T+1	-	0.67	-	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.96
•/• OTH CUR.ASST→COG T+1	1.41	0.90	1.13	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.48
•/• NET PPE→COGS T+1	3.61	3.57	4.31	4.50	4.70	4.85	4.97	5.60	6.00	6.50	6.50	6.50	5.91
•/• OTH ASTS→COGS T+1	0.25	1.88	1.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	1.96
DAYS A/P→COGS T	50.86	90.38	97.41	60.00	60.00	60.00	60.00	60.00	60.00	60.00	60.00	60.00	61.10
DAYS ACCRD.EXP. COGS T	13.05	22.87	21.28	22.00	22.00	22.00	22.00	22.00	22.00	22.00	22.00	22.00	21.90
•/• OTH CUR.LIAB COGS T	7.10	16.25	10.33	15.00	15.00	15.00	15.00	15.00	15.00	15.00	15.00	15.00	14.83
•/• OTH LIAB→COGS T	-	-	-	-	-	-	-	-	-	-	-	-	-
D/E RATIO TO MAINTAIN	-	-	-	30.00	30.00	30.00	30.00	30.00	30.00	30.00	30.00	30.00	28.75
•/• PAYBACK NEW DEBT	-	-	-	12.50	12.50	12.50	12.50	12.50	12.50	12.50	12.50	12.50	12.50
AMORT OF INTANG	-	-	-	-	-	-	-	-	-	-	-	-	-
--NET ADD SKEW FACTORS-													
NAS CT A/RPT DEBT PDOWN	-	-	-	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.15
NAS CT NEW DEBT PDOWN	-	-	-	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50
NAS CT NEW DEBT TAKEON	-	-	-	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50
NAS CT NEW EQUITY TAKEON	-	-	-	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50
NAS TREASURY STOCK	-	-	-	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	-
NAS RETAINED EARNINGS	-	-	-	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.49
NAS CAPITAL EXPENDITURES	-	-	-	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.49
NAS DILUT PRNT DEBT PDWN	-	-	-	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50
	Calculated			Input									

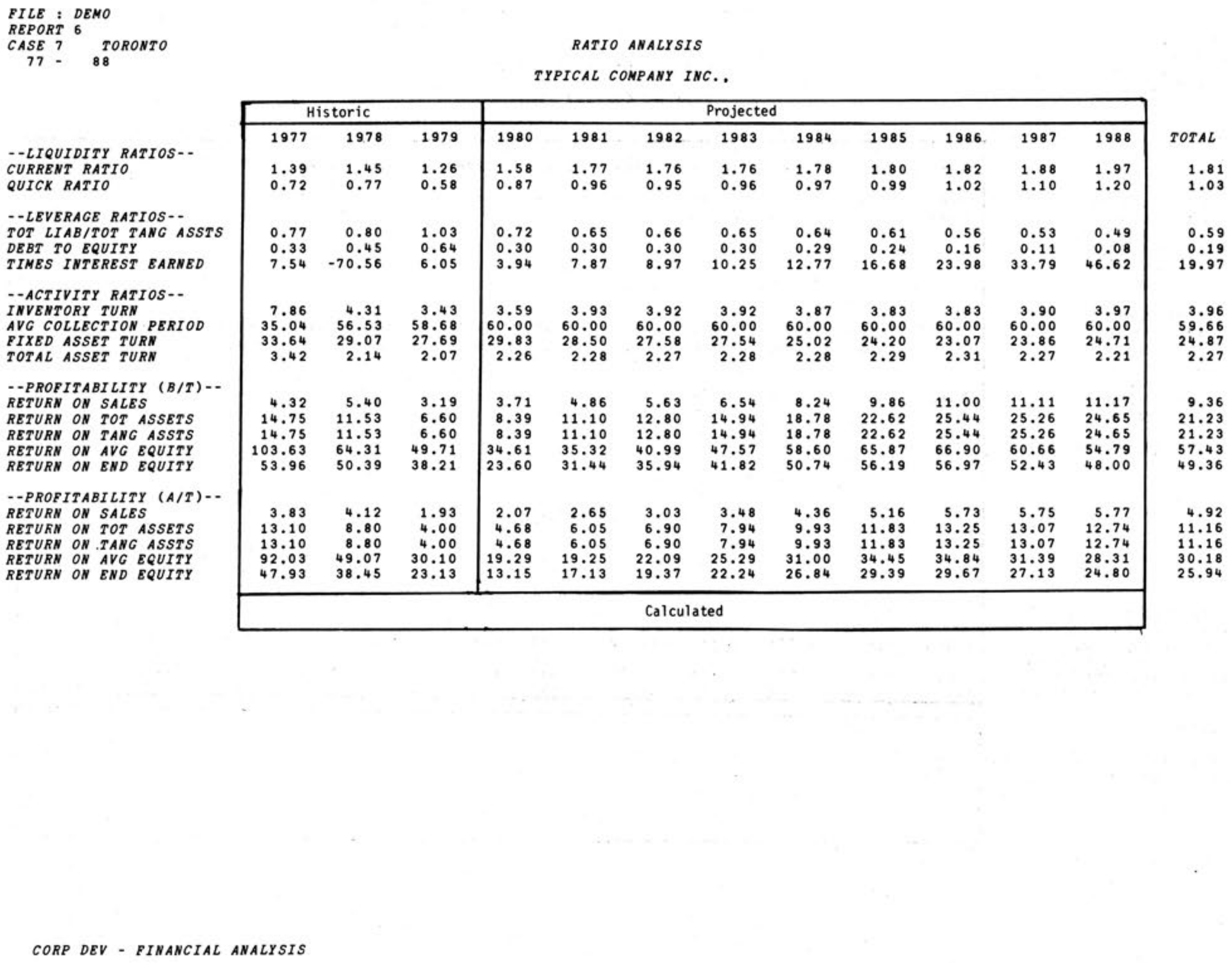
CORP DEV - FINANCIAL ANALYSIS
JUNE 11,1980 11:27

Appendix A — Part 6



```
FILE : DEMO
REPORT 5
CASE 6     TORONTO                                  PRESENT VALUE FORECAST
  80 -    90
                                                     TYPICAL COMPANY INC.,

                              1980    1981    1982    1983    1984    1985    1986    1987    1988    1989    1990    TOTAL
--PRICE--
     UNDISCOUNTED            33185       -       -       -       -       -       -       -       -       -       -    33185
     DISCOUNTED               6923       -       -       -       -       -       -       -       -       -       -     6923

--SALVAGE VALUE--
     UNDISCOUNTED                -       -       -       -       -       -       -       -       -       -   24843    24843
     DISCOUNTED                  -       -       -       -       -       -       -       -       -       -    6141     6141

-CASH EXCESS/(DEFICIT)-
     UNDISCOUNTED             -377    -345    -405    -402    -397     -20     499    1487    2425    3314    3726     9507
     DISCOUNTED               -377    -300    -306    -264    -227     -10     216     559     793     942     921     1947

--PRICE DEDUCTIONS--
     UNDISCOUNTED             1165       -       -       -       -       -       -       -       -       -       -     1165
     DISCOUNTED               1165       -       -       -       -       -       -       -       -       -       -     1165

PRICE/EARNINGS RATIO            22       -       -       -       -       -       -       -       -       -       -       22

                                                           CALCULATED

--ASSUMPTIONS--
--DISCOUNT FACTORS--
     EARNINGS (RRR)          1.000   0.870   0.756   0.658   0.572   0.497   0.432   0.376   0.327   0.284   0.247    0.378
     SALVAGE VALUE (RRR)         -       -       -       -       -       -       -       -       -       -   0.247    0.247
CASH EXCESS/(DEFICIT)        1.000   0.870   0.756   0.658   0.572   0.497   0.432   0.376   0.327   0.284   0.247    0.205
     PRICE DEDUCTION(RRR)    1.000   0.870   0.756   0.658   0.572   0.497   0.432   0.376   0.327   0.284   0.247    1.000

REQUIRED RATE OF RETURN       0.15       -       -       -       -       -       -       -       -       -       -     0.15

FIRST YEAR OF PROJECTION        80       -       -       -       -       -       -       -       -       -       -       80
FIRST MONTH OF DISCOUNT         13       -       -       -       -       -       -       -       -       -       -       13

                                                             INPUT



    CORP DEV - FINANCIAL ANALYSIS
    JUNE 11,1980   12:20 PM
```

Appendix A — Part 7



FILE : DEMO
REPORT 6
CASE 7 TORONTO
77 - 88

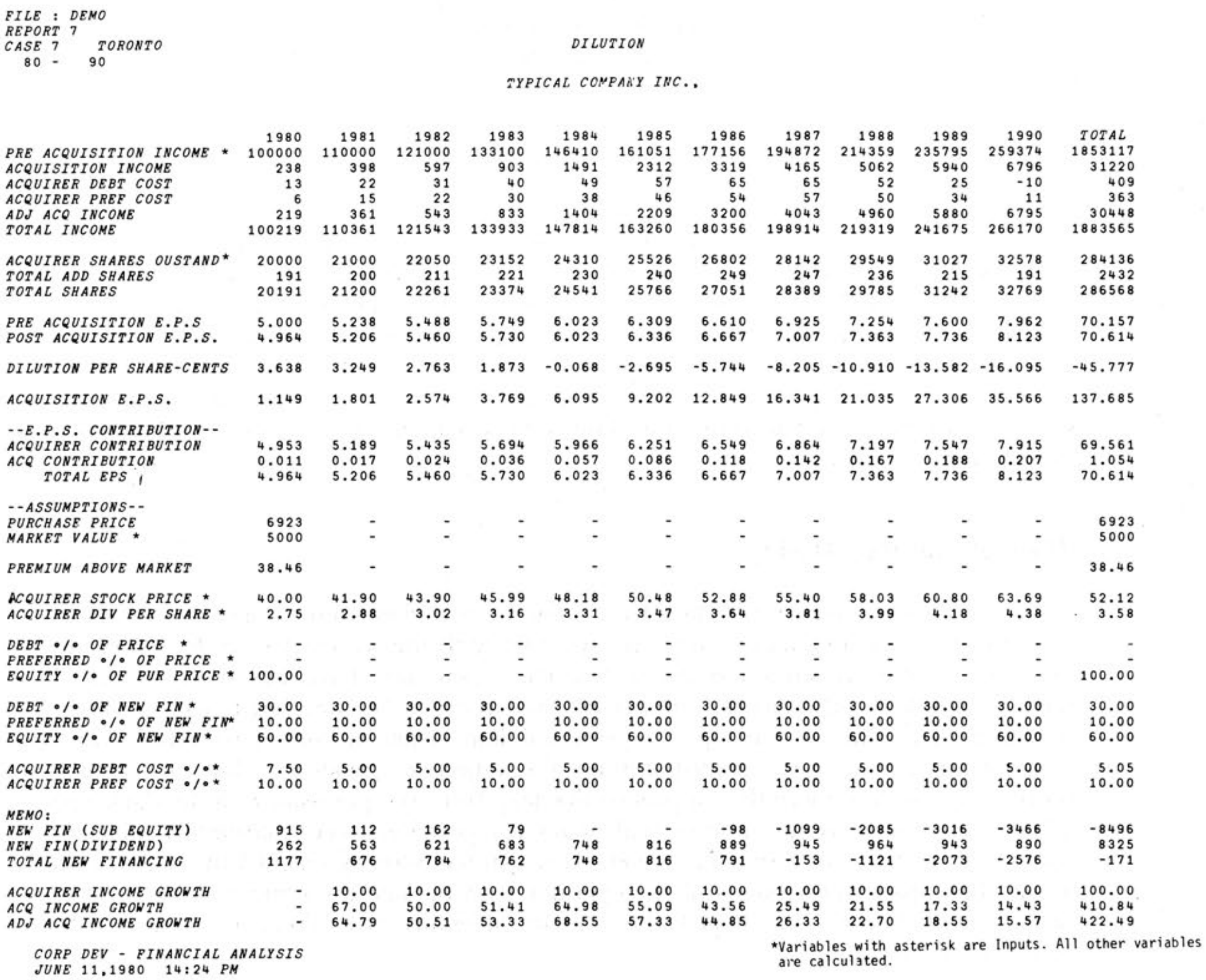
RATIO ANALYSIS

TYPICAL COMPANY INC.,

	Historic			Projected									
	1977	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	TOTAL
--LIQUIDITY RATIOS--													
CURRENT RATIO	1.39	1.45	1.26	1.58	1.77	1.76	1.76	1.78	1.80	1.82	1.88	1.97	1.81
QUICK RATIO	0.72	0.77	0.58	0.87	0.96	0.95	0.96	0.97	0.99	1.02	1.10	1.20	1.03
--LEVERAGE RATIOS--													
TOT LIAB/TOT TANG ASSTS	0.77	0.80	1.03	0.72	0.65	0.66	0.65	0.64	0.61	0.56	0.53	0.49	0.59
DEBT TO EQUITY	0.33	0.45	0.64	0.30	0.30	0.30	0.30	0.29	0.24	0.16	0.11	0.08	0.19
TIMES INTEREST EARNED	7.54	-70.56	6.05	3.94	7.87	8.97	10.25	12.77	16.68	23.98	33.79	46.62	19.97
--ACTIVITY RATIOS--													
INVENTORY TURN	7.86	4.31	3.43	3.59	3.93	3.92	3.92	3.87	3.83	3.83	3.90	3.97	3.96
AVG COLLECTION PERIOD	35.04	56.53	58.68	60.00	60.00	60.00	60.00	60.00	60.00	60.00	60.00	60.00	59.66
FIXED ASSET TURN	33.64	29.07	27.69	29.83	28.50	27.58	27.54	25.02	24.20	23.07	23.86	24.71	24.87
TOTAL ASSET TURN	3.42	2.14	2.07	2.26	2.28	2.27	2.28	2.28	2.29	2.31	2.27	2.21	2.27
--PROFITABILITY (B/T)--													
RETURN ON SALES	4.32	5.40	3.19	3.71	4.86	5.63	6.54	8.24	9.86	11.00	11.11	11.17	9.36
RETURN ON TOT ASSETS	14.75	11.53	6.60	8.39	11.10	12.80	14.94	18.78	22.62	25.44	25.26	24.65	21.23
RETURN ON TANG ASSTS	14.75	11.53	6.60	8.39	11.10	12.80	14.94	18.78	22.62	25.44	25.26	24.65	21.23
RETURN ON AVG EQUITY	103.63	64.31	49.71	34.61	35.32	40.99	47.57	58.60	65.87	66.90	60.66	54.79	57.43
RETURN ON END EQUITY	53.96	50.39	38.21	23.60	31.44	35.94	41.82	50.74	56.19	56.97	52.43	48.00	49.36
--PROFITABILITY (A/T)--													
RETURN ON SALES	3.83	4.12	1.93	2.07	2.65	3.03	3.48	4.36	5.16	5.73	5.75	5.77	4.92
RETURN ON TOT ASSETS	13.10	8.80	4.00	4.68	6.05	6.90	7.94	9.93	11.83	13.25	13.07	12.74	11.16
RETURN ON TANG ASSTS	13.10	8.80	4.00	4.68	6.05	6.90	7.94	9.93	11.83	13.25	13.07	12.74	11.16
RETURN ON AVG EQUITY	92.03	49.07	30.10	19.29	19.25	22.09	25.29	31.00	34.45	34.84	31.39	28.31	30.18
RETURN ON END EQUITY	47.93	38.45	23.13	13.15	17.13	19.37	22.24	26.84	29.39	29.67	27.13	24.80	25.94
	Calculated												

CORP DEV - FINANCIAL ANALYSIS

Appendix A — Part 8



```
FILE : DEMO
REPORT 7
CASE 7     TORONTO                                          DILUTION
  80 -     90
                                                         TYPICAL COMPANY INC..

                                  1980    1981    1982    1983    1984    1985    1986    1987    1988    1989    1990     TOTAL
PRE ACQUISITION INCOME *        100000  110000  121000  133100  146410  161051  177156  194872  214359  235795  259374   1853117
ACQUISITION INCOME                 238     398     597     903    1491    2312    3319    4165    5062    5940    6796     31220
ACQUIRER DEBT COST                  13      22      31      40      49      57      65      65      52      25     -10       409
ACQUIRER PREF COST                   6      15      22      30      38      46      54      57      50      34      11       363
ADJ ACQ INCOME                     219     361     543     833    1404    2209    3200    4043    4960    5880    6795     30448
TOTAL INCOME                    100219  110361  121543  133933  147814  163260  180356  198914  219319  241675  266170   1883565

ACQUIRER SHARES OUSTAND*         20000   21000   22050   23152   24310   25526   26802   28142   29549   31027   32578    284136
TOTAL ADD SHARES                   191     200     211     221     230     240     249     247     236     215     191      2432
TOTAL SHARES                     20191   21200   22261   23374   24541   25766   27051   28389   29785   31242   32769    286568

PRE ACQUISITION E.P.S            5.000   5.238   5.488   5.749   6.023   6.309   6.610   6.925   7.254   7.600   7.962    70.157
POST ACQUISITION E.P.S.          4.964   5.206   5.460   5.730   6.023   6.336   6.667   7.007   7.363   7.736   8.123    70.614

DILUTION PER SHARE-CENTS         3.638   3.249   2.763   1.873  -0.068  -2.695  -5.744  -8.205 -10.910 -13.582 -16.095   -45.777

ACQUISITION E.P.S.               1.149   1.801   2.574   3.769   6.095   9.202  12.849  16.341  21.035  27.306  35.566   137.685

--E.P.S. CONTRIBUTION--
ACQUIRER CONTRIBUTION            4.953   5.189   5.435   5.694   5.966   6.251   6.549   6.864   7.197   7.547   7.915    69.561
ACQ CONTRIBUTION                 0.011   0.017   0.024   0.036   0.057   0.086   0.118   0.142   0.167   0.188   0.207     1.054
    TOTAL EPS                    4.964   5.206   5.460   5.730   6.023   6.336   6.667   7.007   7.363   7.736   8.123    70.614

--ASSUMPTIONS--
PURCHASE PRICE                    6923       -       -       -       -       -       -       -       -       -       -      6923
MARKET VALUE  *                   5000       -       -       -       -       -       -       -       -       -       -      5000

PREMIUM ABOVE MARKET             38.46       -       -       -       -       -       -       -       -       -       -     38.46

ACQUIRER STOCK PRICE *           40.00   41.90   43.90   45.99   48.18   50.48   52.88   55.40   58.03   60.80   63.69     52.12
ACQUIRER DIV PER SHARE *          2.75    2.88    3.02    3.16    3.31    3.47    3.64    3.81    3.99    4.18    4.38      3.58

DEBT •/• OF PRICE  *                 -       -       -       -       -       -       -       -       -       -       -         -
PREFERRED •/• OF PRICE  *            -       -       -       -       -       -       -       -       -       -       -         -
EQUITY •/• OF PUR PRICE *       100.00       -       -       -       -       -       -       -       -       -       -    100.00

DEBT •/• OF NEW FIN *            30.00   30.00   30.00   30.00   30.00   30.00   30.00   30.00   30.00   30.00   30.00     30.00
PREFERRED •/• OF NEW FIN*        10.00   10.00   10.00   10.00   10.00   10.00   10.00   10.00   10.00   10.00   10.00     10.00
EQUITY •/• OF NEW FIN*           60.00   60.00   60.00   60.00   60.00   60.00   60.00   60.00   60.00   60.00   60.00     60.00

ACQUIRER DEBT COST •/•*           7.50    5.00    5.00    5.00    5.00    5.00    5.00    5.00    5.00    5.00    5.00      5.05
ACQUIRER PREF COST •/•*          10.00   10.00   10.00   10.00   10.00   10.00   10.00   10.00   10.00   10.00   10.00     10.00

MEMO:
NEW FIN (SUB EQUITY)               915     112     162      79       -       -     -98   -1099   -2085   -3016   -3466     -8496
NEW FIN(DIVIDEND)                  262     563     621     683     748     816     889     945     964     943     890      8325
TOTAL NEW FINANCING               1177     676     784     762     748     816     791    -153   -1121   -2073   -2576      -171

ACQUIRER INCOME GROWTH               -   10.00   10.00   10.00   10.00   10.00   10.00   10.00   10.00   10.00   10.00    100.00
ACQ INCOME GROWTH                    -   67.00   50.00   51.41   64.98   55.09   43.56   25.49   21.55   17.33   14.43    410.84
ADJ ACQ INCOME GROWTH                -   64.79   50.51   53.33   68.55   57.33   44.85   26.33   22.70   18.55   15.57    422.49

   CORP DEV - FINANCIAL ANALYSIS
   JUNE 11,1980   14:24 PM
```

*Variables with asterisk are Inputs. All other variables are calculated.

USE OF COMMUNICATING WORD PROCESSORS IN NETWORKS

Marvin W. Gaines, Jr.
Lanier Business Products, Inc.
Atlanta, Georgia

Whenever the terms "office of the future" or "office automation" arise in discussions of word processing, invariably communicating word processors are perceived to be an important part of the scenario. Important, yes, but the communicating aspects of these modern typewriting marvels are also perplexing when an attempt is made to explain how best to utilize this equipment for cost effectiveness. In the following, I will present my views on the importance of the communicating word processer in solving both operator resistance to data communications and in enhancing network communications development in the future.

Through the Past Darkly

To begin, we have to turn the clock back a few years and examine a precarious balance. Admittedly, office automation must always start with the secretary's work station, and as long as there are situations where Selectric typewriters have served in the past to input the bulk of office work, automation and information flow begin with a serious handicap. The skill of the typist represented long years of coping tediously with a situation that had masqueraded behind the adjective "modern", but in actuality represented nothing more than a progressive leap from the prehistoric to the dark ages. The editing problems encountered with manual typewriters were marginally alleviated when new methods for disguising the error, which had already reached the paper, came along. The proofreading skills of the operator then became the central issue; the tools were provided; now one hoped the human ingenuity was there to seek out the maladjusted syntax.

Even when attempted corrections had been made, it was time for originators to proofread and restructure thoughts from the past. Soon paragraphs had lost their original context, words were replaced by mysterious synonyms, moments of inspiration gave birth to insertions, and, in the worst of situations, original purposes were revamped, and we began the process again. With this rehashing of thoughts on paper, the tarapin-like speed became the most characteristic of a beast named "turnaround time". The originator's fate to "hurry up and wait" overlooked the word processing operator's plight of growing the third hand which might give aid in producing the fourth, then fifth, revised copy.

Out of this ongoing battle arose "personal dignity": for the operator this came from the satisfaction of a job well done and for the originator it came from the fact that a management relationship was maintained strategically. Though strained at times, the situation, with its curious mixture of peaks and valleys, could be tolerated. In some cases, toleration became the ultimate goal with personal worth becoming entwined with the ability to cope. Promotions were given, not for the ability to manage and take on responsibility, but for the intuitiveness to present one's self worth as an inscrutable commodity not to be tampered with by those seeking production increase. After all, the job did get done.

Broadening Distribution

The product of the office of the past, "black ink on white paper", remains the product of today's office and will maintain its importance in the "office of the future". It is the production speed that we must attack, and as in all wars, strategic planning will best employ our resources. We've seen the first steps where new methods of error correction helped us hide the human factor after it reached the paper, where video screens gave aid to proofreading, and where storage media eliminated tedious rekeyboarding. But how do we improve the distribution of the final copy?

The efficiency of document distribution is only one facet of the entire office automation question. It is true that word processors can be justified solely on the ability to increase productivity and to decrease turnaround time: another way of saying, "You get your work back faster". However, this widely held opinion of word processors is not complete until we consider communicating word processors as a part of the whole solution for the office of the future: another way of saying, "You will get your work faster and further".

No longer will the mails be the only way to distribute documents. No longer will you wait a week for your people in San Francisco to get their notice of a price change from the home office in New York. No longer will you rely on a telephone call to guide your field personnel when hard copy guidelines would do a better job. No longer will checks be cut for field personnel who have long left the field. All this can be accomplished with the assurity that you have time to proofread, to correct, or even to redraft a document and still distribute a hard copy from coast to coast, all within one working day.

Improving Data Communications Through User Education

We are not discussing science fiction nor are we saying "maybe" to the entrance of communications into the office. The time is now. People are evaluating electronic mail because of the tremendous time-saving that could be realized, and in many instances, the transmission facilities for doing that are already in place. However, the main restrictions to such an adaptation are the user's resistance and the exaggerated technical difficulty of converting to applicable systems.

User resistance is created by the fact that such increases in distribution speed are in direct relation to the utilization of "machines", a word of evil to those supposed bastions of human dignity. Personnel acceptance of the new ways of word processing have never been easy. Operators chosen for new equipment cling to their old theories that the personal interaction between originator and word processing support will be destroyed as the ominous keyboards and screens are rolled in.

In most cases, the calming of the operator can only be achieved through proper training. More and more word processing companies have found the solution to be better training methods: manuals written with the complete novice in mind, audio/visual aids supplementing hard copies, and courses directed toward the authority figures within a corporation. In most cases, better results have been produced, but these successes are in the areas of basic word processing not data communications.

What we have is a continuing drama with word processing exemplified as the hero and communications branded a villain. With word processing, the revisions are usually made with handwritten symbols on printed copy, through voices on tapes, or the actual presence of the originator. However, operators do not view the use of table discs, modems, and acoustic couplers as an improvement of human relations; instead, a creeping malaise is produced from the fear that life is becoming more and more complicated. Some will even view the use of phone lines to communicate the product of a typewriter as an incomprehensible aberration. But we try to make it all more palatable. How else can we attribute the heavy reliance on a metaphor like "electronic mail" when explaining the simplest utilization of data communications?

And what goes through the untrained mind when the term "terminal emulation" appears? If taken literally, what do these words call to mind: "a clone of death"? Even if the computer terminal can be conceptualized, all we have is an operator linked into a vast, faceless entity in a distant place never to be seen. The connection between one's everyday correspondence and a computer producing reams and reams of data seems a hard concept to teach, but teach it we must.

The potential of communicating word processors can never be realized until such equipment is more clearly defined. In two specific areas, the word processor can far and away out-perform the terminal. Primarily, the word processor has more programming flexibility available, and secondly the editing capabilities on the word processor are much more powerful than those found on the terminal.

While both the communicating word processor and the computer terminal accomplish the transfer of data to and from a central location, the editing capabilities of the communicating word processor are grossly underrated. The editing capabilities of the terminal are of the secondary importance since the equipment's basic design requires that such changes take place while on-line. As in the case of Lanier equipment, word processors with communicating abilities have an on-line editor mode which allows powerful editing capabilities while maintaining the communications link. In this sense they are faster and, in most instances, cheaper since the link need not be re-established as often.

In coupling both flexibility and editing powers, we must admit that all functions of a word processor, including communications, be "secretary usable". Communications must be taught merely as a function, like sorting or global search, and the fragmentation brought on by the mention of terminals will be eliminated. No longer will it be necessary to learn the peculiarities of two types of equipment, and in centralizing functions we accomplish a silent joining of data processing to word processing. The operator will need only one simple piece of typing equipment for both word processing and communications. Even when the presence of the "vile" computer device is realized it will be too late for a reaction of fear.

The exaggerated technical difficulty stems from the fact that both the novice and the well-informed user realize that the likelihood of there being one vendor for the office of the future is low. So whatever means of data communications which is employed must be able to fit in with the overall office scenario. It is of utmost importance that the modern office have a vital organization with the ability to grow and adapt to change, since both public and private data networks must be utilized for information interchange. Since there will always be different vendors supplying equipment, an interface pattern will always be demanded. The future will bring changes to the present configuration, and the terminal with its interfacing limits will present a dead-end to change. On the other hand, the communicating word processor will change its terminal emulation to match nuances. By simply changing a configuration and reloading, the problem of becoming a broker in differing terminals is avoided.

The Sharp-Lanier Connection

In the particular case of Lanier interfacing with Sharp, a change in the Sharp "Mailbox" program was necessary to allow Lanier-to-Lanier communications through the network, but the change served mainly to simplify Sharp's technical contribution to the project, since the storage area needed to hold messages was only a fraction of the complete network's capabilities. Once the proper control character requirements were provided by Sharp, applicable changes were made on the regular Lanier communications software, and the actual interface was complete.

Within the Lanier organization, all that remained to be done was to coordinate the offices using the network. Twelve locations were designated within the continental U.S., including Atlanta, Chicago, Dallas, Washington D.C., Los Angeles, New York, Seattle, and San Francisco. Three locations were established overseas, including London, Frankfurt, and Sydney. Each Lanier office in the chosen city set up communications on the No Problem and No Problem/XD utilizing the Lanier TELETYPE Communications Option.

The word processing operators in each of these cities received a quick training on the communications options and the Sharp requirements, and preparations were complete. The "Mailbox" requires that two phone calls, one in the morning and one in the afternoon, be made to the Sharp system every day to check for messages for a particular office. The remainder of the day the No Problem is used for regular word processing functions and message transmission. When the operator wishes to type regular correspondence, a "Smart Disc" word processing disc is loaded into the typewriter. When there are communications to be transmitted or received, the TTY-ASCII "Smart Disc" is loaded, a phone call is placed, and messages are stored or collected. If a word processing error is encountered while communicating, the on-line editor mode is engaged with the communications link maintained, and the correction is made.

The types of messages suggested for the system are designed to make economical use of the system since there is a small charge each time a sign-on is initiated. Also, by grouping messages, the cost for each transmission or reception can be stretched further. Though still in the experimental stages, the importance of the system is apparent from the type of correspondence handled:

a) messages to or from the Home Office,
b) unfilled supply orders,
c) payroll information, and
d) urgent parts orders.

As explained earlier, now it is possible for an employee to receive urgent messages while travelling and respond immediately to any situation. Also, a hardcopy of the message is always present for verification.

The actual cost of sending one page of information is less than $1.20, including a sign-on charge and a fixed cost per character. If used properly, your messages can be distributed less expensively than by long distance phone calls or Mailgrams. Of course, the Sharp system is not less expensive than the postal service, but the benefits previously mentioned make for an obvious bargain.

Conclusion

While the issues of saving money and conserving time are the most important goals of the "office of the future", operator simplicity and technical flexibility remain the most prominent hurdles to be overcome. Both of these problems pinpoint the fatal flaws of the computer terminal. The communicating word processor can overcome each, but there is very little true information processing done today to provide a testing ground. Until the office environment can be provided, there will be more talk than action concerning the whole issue of the "office of the future" and how best to include communicating word processors interfaced into networks.

A COMPREHENSIVE SYSTEM FOR PLANNING ECONOMICS

Gregory T. George
The Planning Economics Group, Boston
Woburn, Massachusetts

The history of state-of-the-art software for support of econometric analysis has been one of "turning economists into programmers **without their noticing it**". In the early 1960's, the only recourse available to an economist who wished to use a computer was to become or to ally himself intimately with a very good programmer. Very good programmers have always been in short and unpredictable supply — economists who have employed their services tend to use the words "prima donna" rather than "very good". This was not a stable situation. Fortunately, the last 20 years have seen a large number of rather wonderful improvements in the software available to economic analysts.

What an Economist Does

The prototype problem for an economist in private industry is to predict whether his company's furniture sales will go up or go down. In the public sector, the problem might involve tax revenues, exports, or the unemployment rate of white male economists between 35 and 45 years of age. He is interested in forecasting the behavior of these concepts so that appropriate management actions can be taken — for example, building up a larger inventory of choice hardwoods or postponing the acquisition of a new sanding machine. In other words, he is applying economic analysis to assist his organization in planning. Fortunately, the economy is laced with examples of "lagged dependency". If the economist in our example kept careful track of his sales figures and the residential housing starts in his market area each month, he might find that if housing starts increase or decrease in one month, his sales figures seem to be affected about six months later as new families move into the completed housing and discover that the old sofa just doesn't look right in the new living room. Since some people take longer to make up their minds than others, and since personal disposable income, prospects for employment, and availability of financing might also affect the furniture purchasing patterns of consumers, the economist might wish to analyze the effects of differing lag structures and more independent variables on his ability to predict furniture sales. He might also be interested in sorting out the effects that increased advertising and expenditures in research on changing patterns of consumer taste or on developing new styles of furniture might have on his sales. By the time he starts speculating on whether the steel workers' strike of a few years ago caused people to think better or worse of wooden furniture, he will probably have included so many variables in his analysis that multicollinearity will make it difficult to untangle the contributions of each individual independent explanatory variable from those of the rest.

The next section presents a somewhat subjective view of many of the most important concepts which have been embodied in software to support economic analysis. Although it is organized more or less chronologically, it is intended to familiarize the reader with the general concepts which the Planning Economics Group sought to incorporate into its latest implementation of a comprehensive economic analysis system.

Higher Level Programming Languages

The advent in the early 1960's of what were then considered to be higher level programming languages such as FORTRAN and ALGOL made it possible to record, communicate, and analyze algorithms independently of the context of a particular implementation. The effect was rather analogous — although perhaps somewhat less grand in scale — to the effect of the invention of the moveable type printing press. Suddenly, economists and programmers could communicate in print about the prospects for collecting a sum of squared deviations matrix without employing a room full of graduate students with hundred-key adding machines.

Subroutine Packages

The availability of a **lingua franca** for data analysis algorithms soon led to the dissemination of subroutines to address each of the computational steps involved in such an analysis. If you could get your hands on a subroutine to collect moments, a subroutine to invert a moment matrix, and a subroutine to perform matrix multiplication, you only needed to glue them together with enough FORTRAN mortar to read in the data and print out the results and, **voila**, you had a regression program. Reassemble the building blocks in a little different way and you could perform a correlation analysis. With a little more FORTRAN for mortar, you could try out several different exogenous variables on the right hand side of an equation as predictors of the variable on the left hand side. And if your matrix inversion subroutine began blowing up because of multicollinearity in the data, you had only to find someone who had written a little more robust subroutine and you were back in business again. Analysis of economic data, however, was still firmly in the domain of programming.

Canned Programs for Economic Analysis

By the mid 1960's, the appropriate numerical subroutines had been pretty well debugged and circulated. It became possible to assemble them together into prepackaged or "canned" programs which would perform a particular kind of analysis on data which were presented to them in the appropriate format. Further variations on this theme brought about the "control card" concept, whereby one could provide the image of a FORTRAN FORMAT statement to describe the arrangement of the data to be analyzed and a deck of control cards wherein, for example, a "3" in column 7 might indicate that the analyst wished to suppress the printing of the co-variance matrix for this run.

Language Interpretation and Free Format Data Input

At the middle of the decade, things really began to look up for the non-programmer. Instead of cryptic control cards, economic analysis programs began to speak to the user — or more properly, to **listen** to the user — in his own language. The incorporation of simple language interpreters into software packages freed users from the necessity to specify the format of their data in a FORTRAN-like language, and enabled them to specify data transformations and analysis options in a more natural and mnemonic way.

Systems of Related Functions

Once the analyst could specify options in a more natural way, such as "RESIDUALS=YES" meaning "Yes I do want the residuals to be printed", it became a relatively simple matter to give him words such as FREQUENCIES, CORRELATION, or HISTOGRAM, so that a single program could perform many analyses during one interaction with the user and his data. In this way, the analysis program evolved into the analysis **system** offering one-stop shopping for a variety of data reduction techniques.

The Workspace

Once the economic analysis program became a system capable of performing many functions on a more or less related body of data, it was desirable to have a workspace where the software could maintain the original data and transformed series which were generated from them. With the software keeping track of the whereabouts of the data, the user could communicate his wishes in terms of variable names and observation periods instead of field of locations and record numbers on card, tape, or disk files.

The Time Series Data Type

With the recognition that most economic analysis deals with "time series" data, in which a particular concept is observed at regular intervals such as monthly or quarterly over a number of years, the early 70's saw the development and elaboration of the time series as a data type. The analysis of economic time series is very much easier with a software package which recognizes that it is possible to know a great deal more about a time series than it is a vector of numbers representing a data concept. In addition to the numbers themselves, time series have several useful attributes. They are observed at a particular frequency (e.g., yearly, quarterly, monthly) over a particular period of time (January, 1947 through October, 1980) to a specific degree of accuracy (thousands, millions, units, tenths of a percent) in a particular unit of measurement (dollars, tons, percent) and represent either a stock (the number of tables in inventory on January 1st) or a flow (the number of tables produced during the third quarter of 1980). Other interesting concepts which a program could remember about a time series include the source of the data, the last time they were updated, and the label by which the data should be identified when they appear on a report or a graph.

Immediate (time-shared) and Interactive Execution

A truly major milestone for econometric software was the development of interactive timesharing. First, there were the remote batch systems wherein the user could enter and perhaps edit control statements from a computer terminal and receive printed results back sooner than he might have using the old cards and line printer technology. Then, question and answer dialogues were developed to assist the user in constructing syntactically correct control statements. But these developments did little more than speed up the elapsed time required by the old batch technology. There was very little genuine man-machine interaction. The real breakthrough came when systems became truly interactive so that the course of the analysis could be determined by a sequence of decisions made by the user based on the results of previous interactions. Unlike accounting and record keeping, the analysis of economic data tends to be a tentative, exploratory, and non-deterministic process which benefits greatly from efficient man-machine interaction.

Helpfulness or "User Friendliness"

In very recent times, econometric software has benefited from a growing awareness among software system designers that an investment in improving the helpfulness and sophistication of the man-machine interface can pay significant dividends in user productivity. A specific example is the recognition that the user will probably be using something which looks more like a hard copy or screen terminal than cards and a line printer, and will therefore benefit from output tableaux which are concise and appropriately sized for those devices. In batch mode, it is a good idea to print out every conceivable statistic that can be packed into a 66-line by 132-column line printer page. In interactive mode, the user can be shown the most important part of the results first and given the opportunity to specify whether he wishes to see more supporting detail. An interactive system can provide an editing facility so that the user can immediately recognize and correct any syntax or logical errors in his command specification. In a batch system, it is a good idea to "blow up" and terminate processing immediately, as soon as a command or data error is encountered. In an interactive system — especially one which employs a workspace — the user session should never be terminated involuntarily or in such a way that any work in progress is lost. This is particularly true for economic data analysis where a substantial body of data might be involved in a relatively lengthy trial and error process of investigation.

Interactive Documentation and Error Messages

As economic analysis software systems became more complicated, broader in scope, and truly interactive, the need arose for "on-line" documentation and error diagnostic facilities. These capabilities make it possible for the user to explore somewhat less familiar analysis techniques and to recover from possibly obscure error conditions without involving a programmer or breaking the user's train of thought to consult a lengthy reference manual.

Data Base Management

As economic software systems grew to comprise a set of loosely related analysis functions applied to a loosely related body of data, and as interactive computing reduced the user's need and opportunity to enter his data manually from the keyboard, a major function of the software became managing data bases on disk files. Since time series data tend to be updated on a regular basis, it became important that the software provide a convenient mechanism for storing, retrieving, editing, reviewing, and extending the data. It is also frequently desirable for a number of users to be able to share common storage of widely used data, such as National Income Accounts and common financial statistics.

Modularity and Extensibility

The most comprehensive economic analysis software today incorporates more analysis functions than any individual user can ever hope to become thoroughly familiar with. It is also beyond the scope of any individual programmer to implement and maintain such a system. Yet no system embodies all the analysis functions that the user community might wish to explore. In order to keep up with developments in modelling, econometrics, and numerical analysis, a system must be readily extensible. The most successful extensible systems tend to be modular in construction so that no user or programmer need be familiar with every part of the system, and so that new routines can be added to the system without impacting the behavior of the rest. A user who wants to add a new capability to the system should be able to code that capability in a programming language or in the command language of the system itself, in such a way that it dovetails smoothly with the rest of the system's data management, analysis, and display capabilities.

This list of developments pretty much characterizes the highest state of the art as the 1970's drew to a close. Although no single economic analysis software system did a truly good job of combining all these features, implementation had proceeded to the stage where economists really could perform a very impressive range of time series analyses without feeling that they were programming.

Into the 1980's

Of course, there is always room for improvement. As we looked at the state of economic input to the planning process at the beginning of the 1980's, we saw the need and opportunity for several further developments.

Extensions to Non-time Series Data

Now that the economic time series had been thoroughly conquered and subdued, it became evident that not all interesting concepts in economics come in the form of the simple time series. There is also a great deal to be learned in economics from the analysis of cross-sectional data in which a particular concept, or group of concepts, is observed across a population of, for example, cities and towns, or consumers, or corporations. The observation of cross-sections at multiple points in time leads to the pooled time series/cross-section analysis of two- and higher-dimensional data. In an era when economies are subjected to severe shocks from external political forces, sometimes you can tell more about where you are going by looking around at where others are than by looking back at where you have been.

Eclectic Modelling

The traditional econometric modelling approach involves the estimation and solution of a system of one or more equations which are difference equations on time series observed at regular intervals. This approach tends to be descriptive rather than prescriptive. But there are some parts of an economy, industry, or company which might well be modelled with differential equations, or optimization models, such as linear programming models, or discrete event simulation models, such as queueing models. A good comprehensive economic system for planning economics will admit the possibility of interfacing with and accommodating these kinds of models as well as corporate financial "what if" models and other sources and uses of economic information.

Flexible User Interface and Language

As economic software systems have become more comprehensive, the simple command language structures devised in the late 60's and early 70's have come under some degree of stress. In the middle of adding the next feature into a system, implementers have often discovered that they have already used up the best language construct for that feature in another context. This has led to some rather baroque language constructs with elaborate names and unexpected punctuation cropping up annoyingly. A more helpful and less complicated approach is to divide the language into modules just as the system functions are organized in a modular way. It also turns out that neither question-and-answer, command language, nor menu-driven systems are as useful as a system which combines all three of those techniques for man-machine interaction. The question-and-answer and menu-driven approaches appear to be more structured and, therefore, more useful for novice or infrequent users. The command language approach, which is intimidating to the novice user, is usually quicker and more concise for the experienced user or for simple and frequently used functional areas.

The Use of APL for Economic Analysis

The astute reader may have noticed that many of the desirable attributes of an economic analysis system are applicable to interactive programming systems in general. Furthermore, modern implementations of APL tend to embody very many of these concepts. APL systems are based on a language interpreter which accepts relatively free format data and command input, and maintains functions and data in a workspace. APL file systems provide the ability to maintain data bases on disk files. APL systems are highly interactive, tailored to the characteristics of screen and hard copy terminals, and tend to have had their rough edges sanded down. At least the better APL systems do not blow up and eat your data. APL even provides a programming language in which new capabilities can be implemented in a manner reasonably consistent with the "primitive" system features. APL deals naturally with higher-dimensional arrays of data and has been used to implement a wide diversity of modelling techniques. Thus, it would seem that APL provides an ideal programming environment for interactive economic analysis.

APL as the Implementation Vehicle for a Large Interactive Analysis System

In fact, for the economist who does not mind becoming a programmer, APL may be the ideal vehicle. The APL system shields the user from knowing any more than is absolutely necessary about matters of computer jargon such as Job Control Language, bytes, cylinders, and registers — none of which has any relevance to economic analysis. It provides him with a reasonably suitable set of data structures and functions. Unfortunately, if the economist is to do very much data analysis face-to-face with APL, he will have to become at least an amateur programmer, and will probably notice it. Even if the economist can be brought to overcome the aversion to Greek letters which he gained in graduate school and proceed from the "Symbol Shock" to the "Gee Whiz" phase of learning APL, he must still learn function definition and a host of non-econometric primitives. A great deal can be done to facilitate his use of the APL system by an appropriate choice of functions with which to seed his workspace environment. But in the end there is going to have to be some serious APL programming done. For one thing, APL does not recognize the time series as a primitive data type. APL does not have a built-in facility for establishing a convention for handling data which are missing because of a strike or a tardy government report. APL systems do not tend to be highly regarded for the completeness of their on-line documentation and the helpfulness of their error messages. (Have you ever considered typing ")HELP" after receiving a syntax error message?) Finally, the traditional method of constructing "English language" application systems in APL has involved defining APL functions whose names are evocative of their purpose. If the user then enters the names of the APL functions in an appropriate sequence, he will likely produce a command statement which performs the desired analyses. This simple, trustworthy approach has proved quite adequate for relatively simple systems, but you quickly run out of syntactic glue as the system becomes more comprehensive. First the prepositions and conjunctions are used up: FOR, AND, WITH, BY, TO, with little functions which do nothing more than pass their arguments along to the functions which do the real work. Eventually, the whole thing begins to come apart when you cannot tell whether OVER is a preposition as in "PRINT THE TITLE OVER THE DATA" or an adverb as in "TURN THE PLOT OVER ON ITS SIDE" or "PRINT THE REPORT OVER AGAIN". The most advanced APL implementations, however, contain facilities for overcoming these problems, and offer the fundamentals of a comfortable, high-productivity environment for the developers and users of an interactive system. It was with this idea in mind that The Planning Economics Group set about to implement our next generation of economic analysis software in SHARP APL.

The Tool Bank

We recognized that our system was to be ambitiously large and complex, and that many hands would be involved in implementing, improving, and extending it over a relatively long period of time. We decided that the first need was for a highly organized repository for system code which would enforce modularity, minimize duplicated effort, and keep track of versions, ownership, and inter-dependencies among various parts of the code. We devised what we call a "Tool Bank" which performs all these tasks wCSI=6bV;kJW% for programs and programmers to the APL functions stored within it. Our Tool Bank is an APL file containing functions stored in SHARP APL Packages. The file contains functions for its own maintenance, and provides a place where programmers may store and share additional APL functions. A function invoked by Event Trapping searches the Tool Bank, materializes the appropriate function, and restarts the suspended line or expression, whenever a non-existent "Tool" is referenced. This gives the system the appearance of virtual memory for a very small performance overhead cost. It also prevents the proliferation of multiple copies of commonly used utility functions such as "Vector-To-Matrix" and assures that patches and extensions propagate uniformly and immediately.

Supporting Non-APL Terminals — The Virtual Terminal

Our initial market research indicated that 86.725 percent of all economists and business planners surveyed are completely unfamiliar with APL. We felt it was important for our system to support non-APL terminals, including word- processing systems on which presentation-quality final reports could be produced. We solved this problem by routing all conversation with the user's terminal through input and output functions which use the SHARP APL Arbitrary Input and Arbitrary Output functions. The character set of the user's terminal is mapped into the APL character set used internally in the system. Once we had localized terminal sensitivity, it was possible to code the rest of the system to address a "virtual terminal". Device-specific terminal handling routines are materialized in the user's workspace based on his "Profile" information, which remembers user-specific and session-specific characteristics. In this way we can map paging, windowing, horizontal and vertical scrolling, and transaction memory onto the features and idiosyncracies of a variety of terminals without impacting the system's application modules.

APL as a Subset

On the other hand, there was a great deal of advantage to be gained from allowing users who are sophisticated in APL to adapt and extend the system to accommodate their own needs. Allowing the sophisticated user access to the APL language as a module in the system was not as trivial as it might seem. We wanted to be able to protect the integrity of the system's utility functions and data structures while allowing the APL user as much license as possible. The compromise we finally reached was to split the user's session into a foreground workspace which is connected to the terminal in the normal way, and a background task — running as a SHARP APL N-task — which communicates via Shared Variables. The background task provides a shadow workspace which holds the critical functions and data of our system. Communication between the user and the background workspace, through the Shared Variable interface, is carefully controlled by our system's language processor.

Language and Menu Processor

We were able to obtain enormous flexibility in the user-interface language by implementing a small, table-driven language parser, and a highly automated procedure for generating the parser tables from a higher-level metalanguage description of the system's language. This language processor translates the user's commands from the system's user language into the APL statements which perform the work. For the technically inclined, our language processor is an "Attributed LL(1) Grammar" parser, much like the Finite State Machine parsers common in the literature. It is substantially enhanced by the fact that semantic actions can be expressed in the metalanguage in executable APL. One such action is to switch grammar tables dynamically — invoking a completely new language module. This provides the system with the flexibility to be able to emulate several other popular economic analysis systems, as well as APL itself, without any changes in the bulk of the code comprising the system. It is even possible to tailor the language to the idiosyncracies of a particular user without impacting the system or any other users. Other actions maintain the necessary status information to coordinate menu display and HELP documentation.

Complex Data Types

The major remaining obstacle to an effective system was the need to be able to deal with more complex data types than the Characters, Booleans, Integers, and Real Numbers which are the normal domain of APL. We were able to represent time series and their higher-dimensional analogues, equations, models, etc. as SHARP APL Packages. In this way, informed and cooperating APL functions could communicate by means of a structured bundle of information containing numeric data, character documentation, Boolean information about missing data, and functions for conversion to other representations. Since Packages do not have the rigidity of structure found in the data structures of languages like PL/I or Pascal, it was possible to evolve the Package contents as the system developed.

Delivering the Goods

With all these capabilities in place, there remained the problem of making the system run efficiently. Any very high level language tends to provide the implementer with a great number of opportunities to introduce enormous inefficiencies unknowingly. When a system of considerable size and complexity is put together, large amounts of computer processor time tend to vanish into apparently innocuous procedures. Our experience has shown that there is something approximating an 80-20 rule in operation, where 80 percent of the processor time is spent in 20 percent of the code. The trick is to identify that 20 percent in order to focus your efforts on streamlining the algorithms where they will have the most payoff. Toward this end, we devised a Monitor routine which set Stops on each line of a group of functions, and used Event Trapping to measure the relevant Account Information after the execution of each function line. Although this approach resulted in an overhead during monitoring of several hundred percent, it was sufficiently powerful to enable us to identify the critical lines of code which were the worst offenders. By concentrating our efforts in these places, we have been able to bring the execution costs of the system reasonably well under control with a minimum of "wizard" programming effort.

More to Come

As of this writing, the system is still growing and improving, and we hope and expect that it will continue to do so for some time to come. Current research into directions for the future is concentrated on the application of Artificial Intelligence techniques to make the system appear even more clairvoyant.

PRODUCTION SIMULATIONS IN APL

Charles L. Miller
Johns Hopkins University
Baltimore, Maryland
and
Andrew North
I.P. Sharp Associates Limited
Ottawa, Ontario

Abstract

This paper illustrates the use of simulation and contingency planning by a hypothetical production manager trying to forecast orders and plan production and work force during the 1970 U.S. recession. Our manager adopts scheduling rules first derived by Holt, Modigliani, Muth, and Simon (4,5,8) and "naive" time series forecasting techniques. We examine the sensitivity of decisions to estimates of both cost parameters and orders forecasts. APL — concise of notation and array oriented — serves as a powerful tool.

In Section I, the general nature of the manager's problem is outlined. The use of quadratic functions to approximate relevant costs is sketched in Section II. Section III describes the resulting decision rules. In Section IV, a sensitivity analysis is conducted. Appendix I details the forecast exercise, and Appendix II describes an APL representation of the Holt-Modigliani-Muth-Simon decision rules.

Section I: The Nature of the Decision Problem

Decision-making under conditions of uncertainty is frequently more art than science. Surely this observation is even more accurate if, in addition, the ramifications of current actions extend into an uncertain future. This paper considers a hypothetical production manager who, facing randomness in the arrival of orders, must determine production levels and schedule employment. Not only are adjustments in the decision variables themselves costly, but also forecasting errors cause future readjustments of employment, inventories, and backlogged orders. Thus, the cost consequences of current actions are compounded over time.

The manager's planning problem is three-fold:

(A) **Determine a decision criterion**: The objective is to minimize expected costs within a six-month planning horizon. A mathematical relationship between costs and the decision variables production and employment must therefore be formalized and quantified. Furthermore, the concepts "production" and "employment" must be made operational.

(B) **Derive an optimal policy rule**: "Optimal" here means expected-cost minimizing. The rule specifies — for each six-month forecast of orders and cost parameters (wages, carrying costs, penalty cost for shortages, etc.), and for each initial production rate and work force — a production and employment level.

(C) **Forecast orders**: Because current decisions are based on forecasts which time will doubtless prove incorrect, some provision must be made for forecast error. Decisions judged in retrospect should be deemed good or bad in relation to the information available at the time they were made.

It is typically (A) which forms a moat between real-world business decisions and the embastioned solutions of economic theory. Holt, Modigliani, and Simon (5) built a partial bridge by approximating decision criteria (costs, in this case) with functions quadratic in the decision variables. Their model is founded in factual operating costs: wages, overtime, hiring, lay-off, inventory carrying, and machine setup. Furthermore, they are able to quantify these into a mathematical expected-cost function and derive implementable rules-of-thumb. While their model is specific to the paint factory they consider, their methods extend to a much broader range of decision problems. Because derived rules are based on empirical approximations, the manager must also be concerned with their sensitivity to errors in the estimation of various cost factors.

Once a decision criterion has been established, and the relevant parameters quantified, an optimal rule can be derived using dynamic programming combined with some sort of forecasting methods. Since its earliest development (see Bellman (1)), dynamic programming has found applications in production scheduling and smoothing, inventory control, the management of portfolios and cash balances, and project scheduling, to name but a few. Unfortunately, the technique is rather sophisticated mathematically, and in many cases, fails to yield solutions which can be numerically computed. Holt, Modigliani, and Muth (4) introduced for (B) a method by which a general solution, linear in forecasted orders, is obtained. The optimality of this solution, it should be noted, requires that at each future date forecasts be revised as more information becomes available (see Simon (8) and (1)).

Criteria by which forecasts (C) are judged "good" or "bad", in turn, are sensitive to the choice of decision criteria. The value of information lies not necessarily in its improvement of forecasting accuracy, **per se**, but rather in its reduction of costs by improvement of the decision rule. For our manager, the minimization of expected costs makes it paramount that forecast errors be zero on average. Nonetheless, because the expected cost function is also quadratic in **changes** in the decision variables (see Section II below), large errors, however infrequent, are severely penalized. Using several different contingent forecasts can be useful in reducing the likelihood and expense of such large inaccuracies.

Section II: The Cost Function

In their application of linear decision rules to production and employment scheduling, (5) use the cost function:

$$C_N = \sum_{t=1}^{N} (C_1-C_6)W_t + C_2(W_t-W_{t-1}-C_{11})^2 + C_3(P_t-C_4W_t)^2 + C_5P_t$$

$$+ C_{12}P_tW_t + C_7(I_t-C_8-C_9O_t)^2 + C_{13}$$

where:

W_t = current work force (people)

P_t = aggregate production (units)

I_t = net inventory (units)

O_t = ordered shipments (units)

and, by definition,

$P_t-O_t = I_t-I_{t-1}$ for $t=1,2,\ldots,N$

and $C_1,\ldots,C_{13}$ are constants. C_N represents the total costs of operating over N months. The net change in inventory is defined to be the excess of production over new orders.

The interested reader will find a very lucid development and explanation in (5). Briefly, $(C_1-C_6)W_t + C_2(W_t-W_{t-1}-C_{11})^2$ is intended to capture regular payroll and the costs of hiring, turnover and layoffs. $C_{12}P_tW_t$ gauges the interaction between work force size and production rate. $C_7(I_t-C_8-C_9O_t)^2$ represents inventory carrying costs and run-out penalties. The terms $C_3(P_t-C_4W_t)^2 + C_5P_t$ are meant to capture overtime expenses, varying with work force levels and production rates.

For an actual paint factory, (5) estimated the values displayed in Table I.

Table 1

Cost Coefficients Estimated by Holt-Modigliani-Simon (5)

C_1	C_2	C_3	C_4	C_5	C_6	C_7
340	64.3	0.2	5.67	51.2	281	0.0825

C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}
320	0	59	0	0	0

These numerical values are assumed to describe costs associated with our hypothetical production process as well. As (5) argue, they should typically be derivable from statistical estimates based on both actual accounting data and subjective judgments. Multi-product operations with different types of labour (or other productive factors) can also be approximated using normalizing units.

The adequacy of any such approximation is not goodness-of-fit, but rather the reduction in costs resulting from the derived optimal rule.

Section III: The Optimal Decision Rule

Our manager obtains a decision rule which minimizes the expected value of total costs over a six-month horizon (N=6). The choice of N — a complex problem — is not addressed. It will be sensitive to both forecasting accuracy and the relative magnitudes of adjustment costs versus costs of deviations from optimal inventories.

At the beginning of each month, the manager incorporates available information into several orders forecasts, and then applies the decision rule to determine work force and production. His forecast methods are detailed in Appendix I. This process is repeated each month. To be truly optimal, his rule must take into account the fact that more information will become available by which forecasting accuracy will be improved. Hence it is desirable to postpone decisions for as long as possible. Nonetheless, drastic reactions to current considerations may spawn large, costly future readjustments. The optimal rule recognizes the intertemporal trade-off and tempers reactions accordingly. As (4) show, their rule calls for smooth paths in decision variables which obviate the increasing penalties imposed by the quadratic cost function on large adjustments.

Optimal decision rules for production and work force are reported in (5, eqs. 10 and 11) with weights computed for a twelve-month — rather than our six-month — forecast horizon. Rules are comprised of (1) a weighted average of expected future orders (the quadratic form of costs implies that the distribution of forecast errors is not relevant to expected cost minimization), and (2) a linear combination of the initial levels of work force and inventories. Their reported coefficients are specific to the estimated cost parameters $C_1,\ldots,C_{13}$.

Virtues of the rule described in (5) are its linear form and its independence of the distribution of forecast errors. It provides the production manager with a conveniently updated benchmark against which his actual decisions can be measured, and also can be used to ask a variety of "what if" questions in gauging future material, labour, and capacity requirements. Furthermore, its performance can be measured over time, so that costly inaccuracies can be made less frequent with experience. When cost parameters change, new coefficients can be easily recomputed. Indeed, the APL function *RULE* in Appendix II returns a production and employment choice for an orders forecast of arbitrary horizon and for any set of cost parameters leading to a well-defined expected-cost minimization problem.

Section IV: Simulation Exercises

Two representative types of simulation exercises are now conducted. We first examine the sensitivity of optimal decisions to inaccuracies in the estimation of cost coefficients. We then compare, for the period January 1970 to November 1970, the costs and policies under perfect foresight with those arising from two forecasting procedures:

univariate Box-Jenkins and bivariate transfer function models. In the latter method a survey series — percentage of vendors reporting slower deliveries — is used to forecast orders. These popular methods are chosen for illustrative purposes; other alternatives include causal econometric models and marketing surveys.*

Our orders series, as detailed in Appendix I, is unfortunately seasonally adjusted. This smoothing eliminates the challenging problem of adjusting work force and production levels to more severe seasonal variations in orders. The rule in (5) in fact, predicts that short-term variations will be met chiefly by overtime and production adjustments, while the work force is adjusted only to more enduring trends. This behaviour follows from the costs of lay-off and hiring being quadratic in the cost function. The absence of seasonal fluctuations renders our results less interesting. The methodology, however, is not vitiated.

Each simulation spans the months January to November, 1970. The initial conditions of December, 1969 are assumed to be:

Orders	=	474.5
Employment	=	75
Inventories	=	350
Production	=	530

The orders observation is the actual value of MO/PWMSA ((US manufacturers' seasonally adjusted net new orders, in billions of current dollars) deflated by (seasonally adjusted wholesale price deflator for manufacturing goods), as described in Appendix I). The other values, while fictitious, are nonetheless representative of a plant which, after five years of steady economic growth, is on the verge of a recession. That is, they are higher than those levels at which a perfect foresight manager, using the optimal rule over the period 1965 to 1969, would have arrived. Inventories and production are measured in the same units as orders. Employment is the number of people (choice of hours and overtime is not considered explicitly). We also assume, for convenience, that no orders are backlogged over the simulation period.

BENCHMARK Simulation:

Our first simulation, BENCHMARK, uses the cost parameters reported in Table I and assumes that the true values of future orders were known each month. For example, when the manager planned production and employment at the beginning of

* The predictions arising from these "naive" methods may or may not be "optimal" in, say, the sense of minimizing root-mean-square error. While Box-Jenkins may be best in its restricted class of predictors, problems arise from the use of an estimated cost function. The issue is an important one. Suppose the manager wished to decide how much more money to spend on gathering information and improving forecast accuracy. It appears to be an open question whether he can accurately compute the marginal value (i.e. reduction in costs) of more information (buying, for example, rights to access some economic data bank) or more accurate forecasts (for example, buying forecasts from an econometric firm with a better track record). Simulation is, at least, a sensible first step.

January, 1970, the true values of orders for January through July were known. Thus, to the extent that our cost function accurately measures true production costs, BENCHMARK contains the best possible decisions.

As shown in Table III, our manager gradually decreased employment from its initial level of 75 to 64.9 by November, 1970. Production likewise fell smoothly as the manager contended with the recession-induced decline in orders. Measured operating costs averaged 30007 with little variance over the eleven months.

Exercise I: Variations in Cost Parameters

To what extent will inaccuracies in the measurement of true costs mislead the manager? This is, in general, a difficult statistical question. A first approximation can be obtained by varying the cost parameters, calculating the corresponding decision variables, and then comparing the actual operating costs. Part of the return from an investment in developing optimal rules is that, by simulation, sources of error can be isolated. In this section, the cost function of BENCHMARK is taken as the "true" function.

The simulations DOUBLEW, HALFW, and DOUBLEC7 reflect different cost assumptions than those of BENCHMARK. These assumptions are listed in Table II. Simulation results are reported in Table III.

Simulation Characteristics

Simulation	Cost Assumptions	Forecast Assumptions
BENCHMARK	H-M-S parameters (Table I)	Perfect Foresight
USINGBJ	H-M-S parameters (Table I)	Univariate Box-Jenkins model
USINGTF	H-M-S parameters (Table I)	Bivariate transfer function model
DOUBLEW	Double C_1, C_2 and C_6	Perfect Foresight
HALFW	Halve C_1, C_2 and C_6	Perfect Foresight
DOUBLEC7	Double C_7	Perfect Foresight

Table II

Before comparing the simulations, two points are in order. First, the initial values of December, 1969, were chosen to be consistent with a plant whose cost function is represented by the Holt-Modigliani-Simon (H-M-S) parameters. These hardly need be consistent with the variations, and we must keep in mind the costs of the transition process as inventories and employment are adjusted to stable average levels consistent with the varied cost parameters. Such costs, however, should be considered when parameter estimates are to be revised. If seasonal variations exist in the orders series, these transition costs will be more difficult to measure.

Second, while decision rules are linear in forecasted orders and initial values, they are not linear in all cost parameters. As is typical in dynamic programming — where so many static and intertemporal trade-offs are reflected in optimal rules — intuition about the reaction of decision variables to variations in parameters doesn't go too far. Intuition is further clouded by historical characteristics of the simulation period and myopic forecasting.

Simulation Results

Simulation: BENCHMARK

Period		Employment	Inventories	Production	Costs
January	1970	72.42	352.73	485.50	30768.99
February	1970	70.40	351.19	479.14	30307.73
March	1970	68.83	348.92	474.84	30030.56
April	1970	67.65	353.91	471.68	29877.60
May	1970	66.74	350.18	472.47	30025.14
June	1970	65.99	344.08	472.65	30117.35
July	1970	65.41	340.52	470.73	30010.80
August	1970	65.01	341.87	468.19	29840.81
September	1970	64.79	334.57	467.21	29758.57
October	1970	64.74	344.86	464.26	29529.38
November	1970	64.90	360.23	466.81	29818.15

Simulation: USINGBJ

Period		Employment	Inventories	Production	Costs
January	1970	73.47	373.51	506.28	32251.55
February	1970	71.62	375.29	482.46	30567.91
March	1970	70.05	374.46	476.28	30171.46
April	1970	68.62	376.58	468.80	29718.06
May	1970	67.21	359.27	458.89	28925.64
June	1970	66.54	350.90	470.39	29851.14
July	1970	66.20	351.58	474.98	30299.53
August	1970	65.87	357.48	472.73	30183.43
September	1970	65.34	347.59	464.62	29496.40
October	1970	65.21	366.12	472.50	30327.92
November	1970	64.35	364.76	450.08	28506.78

Simulation: USINGTF

Period		Employment	Inventories	Production	Costs
January	1970	73.34	372.25	505.02	32178.54
February	1970	71.16	366.51	474.95	30022.04
March	1970	69.67	367.58	478.18	30303.91
April	1970	68.33	371.89	470.99	29880.38
May	1970	66.84	352.54	456.85	28778.44
June	1970	66.17	343.94	470.16	29856.77
July	1970	65.95	346.68	477.03	30502.77
August	1970	65.83	358.39	478.55	30724.39
September	1970	65.19	347.16	463.28	29407.91
October	1970	64.83	358.37	465.18	29676.97
November	1970	64.24	363.94	457.01	29091.35

Table III

Simulation Results

Simulation: DOUBLEW

Period		Employment	Inventories	Production	Costs
January	1970	71.21	354.57	487.34	31573.91
February	1970	68.08	353.36	479.47	31033.38
March	1970	65.51	350.60	474.35	30770.36
April	1970	63.44	354.72	470.81	30693.63
May	1970	61.74	349.81	471.28	30970.82
June	1970	60.31	342.36	471.31	31207.07
July	1970	59.17	337.64	469.58	31238.33
August	1970	58.29	338.13	467.32	31185.61
September	1970	57.66	330.23	466.61	31229.70
October	1970	57.24	340.14	463.88	31056.90
November	1970	57.01	355.13	466.43	31449.49

Simulation: HALFW

Period		Employment	Inventories	Production	Costs
January	1970	65.87	454.40	587.17	49926.94
February	1970	57.98	498.92	525.20	44674.94
March	1970	51.88	514.08	492.27	41610.42
April	1970	47.49	523.33	475.94	40360.56
May	1970	44.55	517.48	470.36	39969.13
June	1970	42.63	506.93	468.20	39865.46
July	1970	41.38	499.45	466.82	39881.60
August	1970	40.56	498.31	465.69	40014.25
September	1970	40.06	489.80	466.00	40030.08
October	1970	39.71	500.85	465.03	40366.85
November	1970	39.61	517.92	468.51	41454.77

Simulation: DOUBLEC7

Period		Employment	Inventories	Production	Costs
January	1970	72.49	344.49	477.26	30043.29
February	1970	70.62	339.91	476.11	29946.28
March	1970	69.20	336.71	473.91	29828.57
April	1970	68.15	341.00	470.97	29671.74
May	1970	67.36	338.29	473.50	29962.40
June	1970	66.69	334.09	474.56	30137.00
July	1970	66.15	331.97	472.18	29993.94
August	1970	65.76	333.81	468.67	29737.94
September	1970	65.53	326.88	467.58	29657.17
October	1970	65.47	334.67	461.76	29162.85
November	1970	65.64	347.10	463.87	29365.92

Table III (cont.)

The simulations DOUBLEW and HALFW are based on a doubling and halving, respectively, of the coefficients $(C_1 - C_6)$ **and** C_2 .

DOUBLEW leaves the production series virtually unchanged, but the work force falls to 57 rather than 64 as in BENCHMARK. The cost column in Table III contains "true" costs, i.e. the costs of decisions erroneously made but measured using the H-M-S parameters. They exceed BENCHMARK costs by 2.6 percent in January and grow to a 5.5 percent margin by November. In this simulation, the costs of inaccurate cost parameter estimations are slight.

HALFW increases costs more spectacularly: by an average of 35 percent from July to November, after the transition is made. Several ingredients form this result. Much of the increase can be attributed to the enormous January production surge (587 versus 485 in BENCHMARK) which saddles the firm with enormous inventories in a recession. The decline in orders which further penalizes these holdings in the last months was not anticipated by a six-month perfect forecast made in January. Even though apparent labour costs are lower, erroneous plans compounded by myopic forecasts result in employment being much lower (by 35 percent) by August.

DOUBLEC7 unearths a quirk in the H-M-S assumptions, and signals another problem in simulation design. C7, the coefficient associated with inventory adjustments, is doubled. Not surprisingly, employment and production are virtually unchanged. Inventories average 3 percent less than those of BENCHMARK. Most of this drop occurs in the first month. Unlike HALFW, where historical events penalized a combination of cost inaccuracy and myopic behaviour, in DOUBLEC7 these are rewarded! A reinforcing factor is $C_9 = 0$ so that the firm does not bear added risk of being caught short with smaller inventories. Inventories, then, are lowered at a time when the ensuing recession penalized stocks, so that our experimental increase of the cost parameter provided incentive to do exactly the right thing. Costs average less than those of BENCHMARK.

Exercise II: Costs of Forecasting Techniques

The simulations USINGBJ and USINGTF measure the relative value of the two forecasting methods described in Appendix I. As Table III shows, each increases costs over BENCHMARK, primarily because both react too slowly to the oncoming recession. Thus inventories are too high relative to orders beyond an imperfectly forecast horizon.

Another interesting simulation would be to vary the length of the forecast horizon. Forecasting accuracy, **per se**, is not the measure of desirability of forecasting techniques. Rather, the manager should be concerned with reduction in costs. Simulation may help to balance the trade-off between the costs of myopic planning and the forecasting inaccuracies — common to any method — which rise as the prediction horizon increases.

Table IVa illustrates a quantitative measure of forecasting accuracy, while Table IVb provides estimates of the cost penalties associated with the forecast errors. "True" orders in Table IVa were calculated as current production less change in inventories, initial inventories again assumed to be 350. Values displayed as both univariate and bivariate forecasts are one-month-ahead forecasts based upon the latest available information, i.e. the first entries in each of the columns of Tables V and VI in Appendix I. Negative percent errors indicate forecasts exceeding true orders, reflecting undue optimism on the part of our manager.

The cost figures shown in Table IVb are taken from Table III. Negative percent errors here reflect the percent increase in costs incurred by the manager as a result of inaccuracies in his orders forecasts.

Appendix I: The Forecast Exercise

In an effort to adapt quickly and effectively to a dynamic economic environment, our hypothetical production manager adopts decision rules based upon a combination of cost estimates and anticipated orders. A forecast function for the quantity of orders was generated according to each of two techniques: the univariate time series analysis methods promulgated by Box and Jenkins (2), and the bivariate transfer function models described by the same authors (and discussed in more practical detail in Jenkins (6)).

The Data

The data subjected to the forecast exercise was the quantity Q of new orders, calculated as

$$Q = MO/PWMSA$$

where the series MO represents manufacturing new orders in millions of dollars (seasonally adjusted), and PWMSA the wholesale price index for manufactured goods (base 1967, seasonally adjusted). The series MO and PWMSA were retrieved from the D.R.I. Capsule Data Bank (3). PWMSA simply serves as a deflator, changing the units of measurement from (price × quantity) to quantity. Data from which the seasonal effects had not first been removed might well have proved more illuminating; such data were unfortunately unavailable.

The timeframe for the analysis was selected on the basis of relatively stable activity in the national economy, as reflected in the behaviour of the unemployment rate. A plot of this time series from 1960 to the present suggested the five-year period 1965 through 1969 as the most stable economically; these 60 months constitute the timeframe of the analysis. Our experiments then simulate the manner in which our manager attempts to adapt, on a month-to-month basis, his production and work force schedules to the approaching recession of 1970.

Forecast Errors

		'True'	Univariate	%	Bivariate	%
Period		Orders	Forecasts	Error	Forecasts	Error
January	1970	482.77	503.43	-4.3	505.09	-4.6
February	1970	480.69	489.55	-1.8	480.53	0.0
March	1970	477.11	486.71	-2.0	485.08	-1.7
April	1970	466.69	479.31	-2.7	480.16	-2.9
May	1970	476.20	470.48	1.2	467.25	1.9
June	1970	478.76	476.96	0.4	474.55	0.9
July	1970	474.29	477.96	-0.8	475.63	-0.3
August	1970	466.84	475.98	-2.0	480.72	-3.0
September	1970	474.51	470.20	0.9	474.07	0.1
October	1970	453.97	475.24	-4.7	463.44	-2.1
November	1970	451.44	458.15	-1.5	462.50	-2.4

Table IVa

Cost Penalties Due to Forecast Errors

		BENCHMARK	USINGBJ	%	USINGTF	%
Period		Costs	Costs	Error	Costs	Error
January	1970	30769	32252	-4.8	32179	-4.6
February	1970	30308	30568	-0.9	30022	0.9
March	1970	30031	30171	-0.5	30304	-0.9
April	1970	29878	29718	0.5	29880	0.0
May	1970	30025	28926	3.7	28778	4.2
June	1970	30117	29851	0.9	29857	0.9
July	1970	30011	30300	-1.0	30503	-1.6
August	1970	29841	30183	-1.1	30724	-3.0
September	1970	29759	29496	0.9	29408	1.2
October	1970	29529	30328	-2.7	29677	-0.5
November	1970	29818	28507	4.4	29091	2.4

Table IVb

Univariate Time Series Analysis

The series Q measuring quantity of new orders was first subjected to the three-stage procedure of Box and Jenkins. First-order differencing was required to induce stationarity. As was to be expected, no readily identifiable seasonal component manifested itself. No transformation was considered appropriate.

Model identification and preliminary estimation indicated that a non-seasonal ARIMA model of type (2,1,0) or (2,1,1) might prove representative of the behaviour of Q. However, nonlinear least squares estimation of the three parameters in the second model resulted in significantly correlated parameter estimates, and in particular an estimate of the moving average parameter θ which was disconcertingly small relative to its standard error. This suggested that the moving average parameter was not significantly different from 0, and that the (2,1,0) model seemed more appropriate.

Nonlinear estimation of this model resulted in rapid convergence to stable, uncorrelated parameter estimates, each of which was pleasingly large in magnitude relative to its standard error. Application of diagnostic checks to the estimated residuals provided no indication of model inadequacy.

The exact form of this model was

$$(1 - B)(1 + .2588B + .2245B^2)z_t = a_t$$

where z_t represents a value of the series Q observed at period t, and B is the backshift operator such that $B^k z_t \leftrightarrow z_{t-k}$. a_t denotes the residual, or noise, or random shock at period t.

Note that the generation of forecasts based upon this model is equivalent to the application of a weighted moving sum to the past three observations. Multiplication of the two left-hand side polynomials in B and subsequent collection of terms yields a forecast function of the form

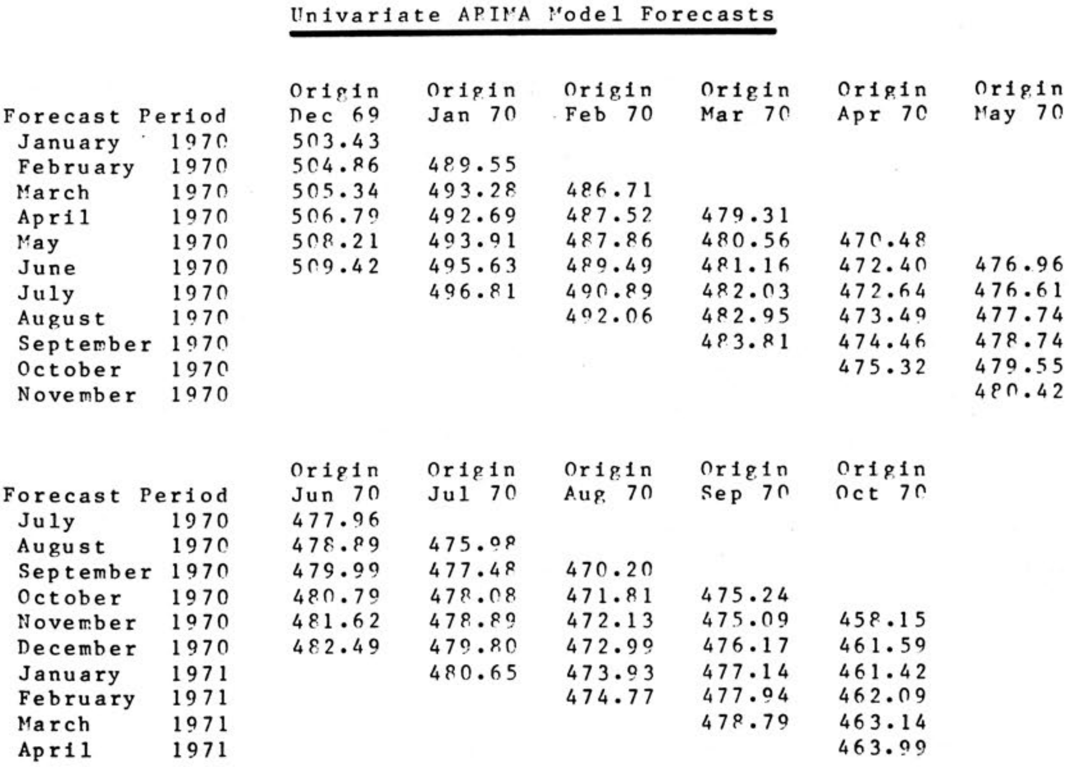
$$z_t = a_t + .74z_{t-1} + .04z_{t-2} + .22z_{t-3}$$

Note also that the weights accorded the observations in the three preceding periods sum to unity.

A succession of updated 6-month forecasts, from forecast origins December, 1969 through October, 1970, is shown in Table V.

A new set of forecasts was generated as each additional value of Q became available. It was assumed that as this new data became known, the type of model describing the series would not be affected significantly, but that the particular values of the parameters might be data sensitive. Model parameters were therefore re-estimated based upon data through March, 1970, and again based upon data through June, 1970. Parameter estimates employed in the generation of the forecast functions were then:

Forecast origin	ϕ_1	ϕ_2
Dec 1969 - Feb 1970	-.2588	-.2245
Mar 1970 - May 1970	-.2018	-.1448
Jun 1970 - Oct 1970	-.2036	-.1522



Univariate ARIMA Model Forecasts

Forecast Period	Origin Dec 69	Origin Jan 70	Origin Feb 70	Origin Mar 70	Origin Apr 70	Origin May 70
January 1970	503.43					
February 1970	504.86	489.55				
March 1970	505.34	493.28	486.71			
April 1970	506.79	492.69	487.52	479.31		
May 1970	508.21	493.91	487.86	480.56	470.48	
June 1970	509.42	495.63	489.49	481.16	472.40	476.96
July 1970		496.81	490.89	482.03	472.64	476.61
August 1970			492.06	482.95	473.49	477.74
September 1970				483.81	474.46	478.74
October 1970					475.32	479.55
November 1970						480.42

Forecast Period	Origin Jun 70	Origin Jul 70	Origin Aug 70	Origin Sep 70	Origin Oct 70
July 1970	477.96				
August 1970	478.89	475.98			
September 1970	479.99	477.48	470.20		
October 1970	480.79	478.08	471.81	475.24	
November 1970	481.62	478.89	472.13	475.09	458.15
December 1970	482.49	479.80	472.99	476.17	461.59
January 1971		480.65	473.93	477.14	461.42
February 1971			474.77	477.94	462.09
March 1971				478.79	463.14
April 1971					463.99

Table V

Choice of Leading Indicator

The second method of forecasting new orders was a bivariate transfer function model (2, Chapters 10 and 11) postulating that changes in Q tend to be anticipated by changes in some leading indicator series. Two possible candidates for an appropriate leading indicator are the series IVPAC and DLEADT from the D.R.I. Capsule Data Bank. IVPAC is a vendor performance indicator, measured as the percentage of companies reporting slower deliveries (**not** seasonally adjusted), while DLEADT is a composite index of twelve leading indicators, including IVPAC.

As an assessment of the relative appropriateness of each of these series as a leading indicator for Q, correlations were measured between levels of each indicator and various lags of Q. All correlations at low lags were significant and of the proper sign (positive: the slower the delivery, for example, the more rapid the pace of economic activity, and therefore the more units one would expect to be ordered). Average durations of runs (successive month-to-month changes of the same sign) were then measured for each series in an attempt to judge their ability to reflect the business cycle. Again, little conclusive evidence was provided.

IVPAC was eventually selected, because vendor performance actually measures a subset of order activity (hence we maintain a presumption concerning its utility as an indicator), and because of difficulties arising from the composite nature of DLEADT. (For example, three of its twelve components have been smoothed in some fashion, two of the twelve are not seasonally adjusted, etc.)

Since Q is measured in seasonally adjusted units, the seasonal component of IVPAC was removed before it was submitted as an input to the transfer function model. This removal was affected according to the X-11 Census Method II Seasonal Adjustment algorithm (7).

Bivariate Transfer Function Model Analysis

The analysis of a bivariate transfer function model proceeds according to the same identification, estimation and forecasting sequence as is employed in a univariate ARIMA analysis. However, the identification process is considerably simplified if the input to the system is white noise. When the original input in fact follows some other stochastic process, simplification is made possible by "prewhitening". Prewhitening is designed to transform the correlated input series to an uncorrelated white noise series. This prewhitening transformation is in effect a univariate ARIMA model representing the behaviour of the input series.

The first step in the transfer function analysis of our production manager, then, was the development of a univariate ARIMA model adequately descriptive of the input series IVPAC (the leading indicator). As was the case with the (output) series Q, no preliminary transformation seemed suitable, and first-order differencing was necessary to induce stationarity. Again, as was to be expected, no readily identifiable seasonal fluctuations manifested themselves. Neither the autocorrelation function nor the partial autocorrelation function contained a single significant element in 24 lags, and no pattern was immediately discernible in the plot of either function. A model of the form

$$(1 - B)z_t = (1 - \theta B^4)a_t$$

was dubiously suggested, but was rejected when subsequent model estimation yielded an estimate for θ which was discouragingly small relative to its standard error. It was decided, therefore, that (del)IVPAC was already a white noise series, and that simple first-order differencing constituted the necessary prewhitening transformation.

The second step in a bivariate transfer function analysis is the identification of the transfer function model **and** the stochastic noise model. Note that influences other than the (single) input series will invariably affect the output series, even under carefully controlled conditions. The combined effect on the output series of these other influences constitutes the disturbance, or noise. Any model which purports to describe an association involving real data must account for not only the dynamic relationship between the input and the output, but also for the noise infecting the system. Such a joint model is obtained by combining a deterministic transfer function model with a stochastic noise model.

Examination of the estimated autocorrelation and cross correlation functions of the output series Q and the leading indicator series IVPAC revealed that first-order differencing was required to induce stationary behaviour in each series. This was already known, as a result of previous analyses, as was the fact that no preliminary transformation was necessary, and that neither series contained a significant seasonal component. Investigation of patterns in the generated impulse response weights suggested that a transfer function model with parameters (r,s,b) equal to (2,0,2) might be tentatively entertained as being representative of the relationship between the two series. (Note that, as discussed in (2) and (6), r=2 implies that the current level of Q is related to the two previous values of Q. s=0 implies that the current level of Q is also related to no values of the input series IVPAC **other than** the value of IVPAC at lag b, the delay parameter. A value of 2 for b indicates that there is one whole period of delay before a change in the level of the input series manifests itself in a change in the level of the output series Q. The current level of output is **always** related to the level of input at lag b.) Examination of the autocorrelation and partial autocorrelation functions of the noise series indicated that an autoregressive model of order 2 was worthy of being entertained as the noise model.

Thus, transfer function/noise model identification suggests a tentative model of the form

$$(1 - B)Q_t = \frac{\omega_0}{(1 - \delta_1 B - \delta_2 B^2)} (1 - B)\mathrm{IVPAC}_{t-2} + \frac{1}{(1 - \phi_1 B - \phi_2 B^2)} a_t$$

where the first term on the right-hand side constitutes the transfer function model and the second term the noise model. (Again, B denotes the backshift operator.) Simultaneous nonlinear least squares estimation of the five parameters in this model resulted in the eventual decline to zero of the second autoregressive parameter ϕ_2 in the noise model. In the interests of the Box-Jenkins "principle of parsimony" this parameter was deleted from the noise model. Subsequent estimation resulted in rapid convergence (3 iterations) to stable, uncorrelated parameter estimates. Only the (single) autoregressive parameter in the noise model was small relative to its standard error, but the combined model was adjudged pleasantly parsimonious as it stood, and ϕ was retained. No further simplification by factorization appeared possible. Examination of the residual autocorrelation function and of the cross correlation function between the estimated residuals and the prewhitened input provided no evidence of inadequacy in either the combined transfer function/noise model or in the transfer function model itself.

The final fitted model selected, and that model based upon which forecasts were ultimately generated, was

$$(1 - B)Q_t = \frac{-.36}{(1 + .97B + .97B^2)} (1 - B)\mathrm{IVPAC}_{t-2} + \frac{1}{(1 + .1B)} a_t$$

Decomposition of this combined model into a transfer function model and a separate noise model, and subsequent multiplication of polynomials and collection of terms, reveals that the current level of quantity ordered is then related to orders in the **three** preceding periods, the level of the leading indicator **two** and **three** periods in the past, and the **current** and **previous** disturbance terms.

A succession of updated 6-month forecasts, from forecast origins December, 1969 through October, 1970, is shown in Table VI.

Bivariate Transfer Function Model Forecasts

Forecast Period		Origin Dec 69	Origin Jan 70	Origin Feb 70	Origin Mar 70	Origin Apr 70	Origin May 70
January	1970	505.09					
February	1970	499.40	480.53				
March	1970	500.51	483.80	485.08			
April	1970	504.48	485.49	486.86	480.16		
May	1970	501.37	482.29	483.58	478.74	467.25	
June	1970	499.05	482.31	483.60	477.15	465.65	474.55
July	1970		485.91	487.28	480.50	468.93	470.94
August	1970			482.85	478.21	466.63	475.43
September	1970				479.14	467.65	476.61
October	1970					466.52	468.71
November	1970						475.31

Forecast Period		Origin Jun 70	Origin Jul 70	Origin Aug 70	Origin Sep 70	Origin Oct 70
July	1970	475.63				
August	1970	481.16	480.72			
September	1970	480.95	485.00	474.07		
October	1970	473.17	472.81	462.26	463.44	
November	1970	481.25	480.88	468.97	470.31	462.50
December	1970	481.38	486.20	474.28	475.48	470.20
January	1971		474.63	464.06	465.25	457.40
February	1971			468.20	469.54	461.69
March	1971				476.92	471.59
April	1971					456.26

Table VI

Again, the forecast function was updated as each additional observed value of Q and IVPAC became available. The form of the forecast function was assumed unaffected by the increased data availability, but the individual parameter estimates were suspected to be data sensitive. The combined transfer function/noise model of the form outlined above was therefore re-estimated based upon data available through March, 1970, and again based upon data available through June, 1970. Parameter estimates employed in the generation of the forecast functions were then:

Forecast origin	δ_1	δ_2	ω_0	ϕ_1
Dec 1969 - Feb 1970	-.966	-.9745	-.3638	-.1037
Mar 1970 - May 1970	-.9756	-.9741	-.3327	-.0911
Jun 1970 - Oct 1970	-.9867	-.9743	-.3325	-.0918

Appendix II: APL as a Simulation Tool

All statistical operations and simulations were performed using APL. Its interactive nature, amenability to mathematical formulae, and facile array handling make it a very powerful tool. While the function *RULE* appears rather complex, the functions used to conduct simulations are quite simple, and could be written by a neophyte. We believe that managers at all levels could profit from the use of simulation techniques, and further have found in our own experience that APL is ideally suited to such experimental exercises. Large-scale exercises, with decision variables ranging into the hundreds, are admittedly awkward (one would be hard-pressed to code and solve a large-scale macroeconomic model containing 800 equations, for example). Smaller, more common models, however, can be coded by almost any user.

The APL functions employed in our simulation exercise are listed below. Liberal use of commentary and references to the literature make them, we believe, self-explanatory to the interested reader. The seasonal adjustment, correlation, and univariate and bivariate time series analyses discussed in Appendix I were performed using functions resident in the SHARP APL public libraries.

```
     ∇ R←C COST∆FUNCTION X
[1]   ⍝ COMPUTES THE COST FUNCTION C[N] GIVEN BY EQUATION (1.1)
[2]   ⍝ OF HOLT, MODIGLIANI AND MUTH (4).
[3]   ⍝ X: MATRIX OF 5 COLUMNS CONTAINING W,I,P,B,O RESPECTIVELY
[4]   ⍝      B IS BACKLOGGED ORDERS, ASSUMED 0 THROUGHOUT.
[5]   ⍝ C: VECTOR OF 13 COST PARAMETERS (SEE TABLE I)
[6]   ⍝ R: VECTOR OF LENGTH (1↑⍴X) CONTAINING COSTS FOR PERIODS T=2,...,N, THEN SUM.
[7]    R←(-/C[1 6])×1↓X[;1]
[8]    R←R,[1.5] C[2]×((-/(1↓X[;1]),[1.5] ¯1↓X[;1])-C[11])*2
[9]    R←R,C[3]×(1↓X[;3]-C[4]×X[;1])*2
[10]   R←R,C[13]+1↓X[;3]×C[5]+C[12]×X[;1]
[11]   R←R,C[7]×(1↓X[;2]-C[8]+C[9]×X[;5])*2
[12]   R←+/R
[13]   R←R,+/R
     ∇
```

```
     ∇ R←COEFFICIENTS∆FROM R
[1]   ⍝ COMPUTES THE DERIVED COST COEFFICIENTS C[14] THROUGH C[23]
[2]   ⍝ FOR THE HOLT-MODIGLIANI-MUTH-SIMON MODEL.
[3]    R←R,(×/2,R[3 4])-R[12] ⍝ P. 162
[4]    R←R,2×R[2]÷R[14] ⍝ P. 162
[5]    R←R,2×R[3]×(R[4]*2)÷R[14] ⍝ P. 162
[6]    R←R,R[3]×R[15]÷R[7] ⍝ P. 163
[7]    R←R,(R[3]×R[16]÷R[7])-R[14]÷2×R[7] ⍝ P. 163
[8]    R←R, 2 1 3 1 +.×R[15 16 17 18] ⍝ P. 164
[9]    R←R, 1 3 1 +.×R[15 17 18] ⍝ P. 164
[10]   R←R, 1 4 1 +.×R[15 17 18] ⍝ P. 164
[11]   R←R, 2 1 6 2 +.×R[15 16 17 18] ⍝ P. 164
[12]   R←R,R[16]+2×R[15] ⍝ P. 162
     ∇
```

```
    ∇ D←C RULE F;S;RADICAL;LAMBDA;A;B;RHO;PHI
[1]  ⍝ COMPUTES OPTIMAL PRODUCTION AND WORK FORCE
[2]  ⍝ ACCORDING TO HOLT, MODIGLIANI AND MUTH (4).
[3]  ⍝ F: VECTOR OF INITIAL VALUES AND ORDERS FORECASTS, SUCH THAT:
[4]  ⍝    F[1] ↔ INITIAL WORK FORCE
[5]  ⍝    F[2] ↔ INITIAL INVENTORIES
[6]  ⍝    2↓F  ↔ ORDERS FORECAST OVER ((⍴F)-2)-PERIOD HORIZON
[7]  ⍝ C: VECTOR OF COST PARAMETERS ASSOCIATED WITH EQUATION (1.1),
[8]  ⍝    SUCH THAT  13↔⍴C  AND  C[10]↔C[1]-C[6].
[9]  ⍝ D: VECTOR CONTAINING DECISION VALUES EMPLOYMENT, PRODUCTION.
[10] ⍝ NOTE: THE CONDITIONS  C[2 3 4 7]>0  AND  0≤C[12]≤4×C[3]×C[4]
[11] ⍝       ARE SUFFICIENT TO ENSURE AN INTERIOR MINIMUM.
[12] ⍝ COMPUTE THE DERIVED COEFFICIENTS C[14] THROUGH C[23]:
[13]  C←COEFFICIENTS∆FROM C
[14] ⍝ DETERMINE WHETHER THE ROOTS OF AUXILIARY EQUATION (3.4)
[15] ⍝ ARE REAL OR COMPLEX:
[16]  →(0>RADICAL←((+/C[15 18])*2)-×/4,C[16 17])⍴COMPLEX
[17] ⍝ COMPUTE THE REAL ROOTS OF THE AUXILIARY EQUATION:
[18] REAL:S←(÷2×C[17])×(+/C[15 18])+ 1 ¯1 ×RADICAL*0.5
[19]  LAMBDA←0.5×(4⍴2+S)+, ¯1 1 ∘.×(S×4+S)*0.5
[20]  LAMBDA←(1>|LAMBDA)/LAMBDA
[21] ⍝ COMPUTE THE REAL COEFFICIENTS OF SYSTEM (3.6):
[22] ⍝ LEFT-HAND SIDE OF (3.6):
[23]  A←((C[19 21 17]× 1 ¯1 1)+.×⍉LAMBDA∘.* 0 1 2),[1.5]-(-/C[20 21])+C[17]÷LAMBDA
[24] ⍝ FIRST 2 EXPRESSIONS ON RIGHT-HAND SIDE OF (3.6)
[25] ⍝ (WEIGHTED SUM OF ORDERS FORECASTS):
[26]  B←(1+C[9]×1-LAMBDA)×(LAMBDA∘.*¯1+⍳⍴2↓F)+.×2↓F
[27] ⍝ RIGHT-HAND SIDE OF (3.6):
[28]  B←+/B,[1.5]((-F[2])+C[8]-(÷/C[10 14])÷1-LAMBDA)+F[1]×C[15]+C[17]×1-LAMBDA
[29]  →OUT
[30] ⍝ COMPUTE THE ROOTS OF THE AUXILIARY EQUATION AND SEPARATE
[31] ⍝ THEM INTO THEIR REAL AND COMPLEX PARTS
[32] ⍝ (NOTE: NOTATION CORRESPONDS TO THAT OF PAPER):
[33] COMPLEX:S←(÷2×C[17])×(+/C[15 18]),[1.5] 1 ¯1 ×(|RADICAL)*0.5
[34]  RADICAL←(S[1;] CMULT 4 0 CADD S[1;]),[0.5] S[2;] CMULT 4 0 CADD S[2;]
[35]  R←2 NORM RADICAL[1;]
[36]  RADICAL←(÷2*0.5)×((R+RADICAL[1;1])*0.5),[1.5] 1 ¯1 ×(R-RADICAL[1;1])*0.5
[37]  LAMBDA←0.5×(S+(⍴S)⍴ 2 0)-RADICAL
[38]  RHO←2 NORM LAMBDA[1;]
[39]  PHI←¯3○÷/⌽LAMBDA[1;]
[40]  LAMBDA←RHO×(2○PHI),[1.5] 1 ¯1 ×1○PHI
[41] ⍝ COMPUTE THE REAL AND COMPLEX COEFFICIENTS OF SYSTEM (3.6):
[42]  A←(C[19],0)+((-C[21])×LAMBDA[1;]) CADD C[17]×LAMBDA[1;] CMULT LAMBDA[1;]
[43]  A←A,[1.5] C[17]× 1 0 -CRECIP LAMBDA[1;]
[44]  B←(-F[2],0)+(C[8],0)-(÷/C[10 14])×CRECIP 1 0 -LAMBDA[1;]
[45]  B←B+F[1]×(C[15],0)+C[17]× 1 0 -LAMBDA[1;]
[46]  B←B+(1 0 +C[9]× 1 0 -LAMBDA[1;])×(((2○PHI×(⍳⍴2↓F)-1),[0.5] 1○PHI×(⍳⍴2↓F)-1))+.×(RHO*¯1+⍳⍴2↓F)×2↓F
[47] OUT:
[48] ⍝ COMPUTE OPTIMAL WORK FORCE AND PRODUCTION RATE.
[49] ⍝ SOLVE FOR W1 AND W2 ACCORDING TO (3.6):
[50]  D←B⌹A
[51] ⍝ SOLVE FOR PRODUCTION ACCORDING TO (2.5):
[52]  D←D[1],(÷/C[10 14])+(C[15 23 15]× ¯1 1 ¯1)+.×F[1],D
    ∇

    ∇ R←A CADD B
[1]  ⍝ ADDS TWO COMPLEX NUMBERS
[2]   R←+/A,[1.5] B
    ∇

    ∇ R←A CMULT B
[1]  ⍝ MULTIPLIES TWO COMPLEX NUMBERS
[2]   R←(-/×/A,[1.5] B),+/×/A,[1.5]⌽B
    ∇

    ∇ R←CRECIP A
[1]  ⍝ COMPUTES THE RECIPROCAL OF A COMPLEX NUMBER
[2]   R←(1 ¯1 ×A)÷+/A*2
    ∇

    ∇ R←N NORM X
[1]   R←(+/X*N)*÷N
    ∇
```

References

(1) Bellman, Richard, **Dynamic Programming**, Princeton University Press, Princeton, N.J., 1957.

(2) Box, G.E.P. and G.M. Jenkins, **Time Series Analysis: Forecasting and Control**, Revised Edition, Holden-Day Inc., San Francisco 1976.

(3) **D.R.I. Capsule Data Bank Directory**, Data Resources, Inc.

(4) Holt, Charles C., Franco Modigliani and John F. Muth, "Derivation and Computation of Linear Decision Rules for Production and Employment Scheduling", **Management Science**, Volume 2, pp. 159-177, 1956.

(5) Holt, Charles C., Franco Modigliani and Herbert A. Simon, "A Linear Decision Rule for Production and Employment Scheduling". **Management Science**, Volume 1, pp. 1-30, 1955.

(6) Jenkins, G.M., **Practical Experiences with Modelling and Forecasting Time Series**, Gwilym Jenkins and Partners Ltd., 1979.

(7) Shiskin, J., Allan H. Young and John C. Musgrave, **The X-11 Variant of the Census Method II Seasonal Adjustment Program**, Technical Paper 15, Bureau of the Census, U.S. Dept. of Commerce, Revised February 1967.

(8) Simon, Herbert A., "Dynamic Programming Under Uncertainty with a Quadratic Criterion Function", **Econometrica**, Volume 24, pp. 74-81, 1956.

AN INFORMATION SYSTEM FOR FLEET MANAGEMENT

George A. Clark
Potash Corporation of Saskatchewan
Saskatoon, Saskatchewan

Introduction

The railway industry in North America operates a fleet of about two million railcars over 370,000 miles of track. With an operation of this magnitude, it is not surprising that railcars are delayed or misappropriated on a regular basis. Cars are "lost" in railway yards, left on sidings to be picked up at the railway's convenience, or "accidentally" assigned to the wrong shipper. Receivers hold cars at the destination, using them for storage as well as transportation, and repair shops allow cars that are owned by small or uncomplaining customers to sit on tracks, while they give priority to equipment that belongs to more demanding or more "important" customers.

At one time, virtually all railway cars were owned by the railway industry. This has changed dramatically, to the point where the railways now own only about 80 percent of railway equipment. Since railroad investment in railcars has not kept pace with traffic growth, many shippers have seen no alternative but to lease or buy cars for their own use. There are many opportunities for delay or misuse of railway cars, and once a shipper has a fleet of his own he is well-advised to exert tight control over the use and movement of his cars.

In fact, there is a powerful financial incentive to increase the use of leased or privately-owned railcars. The railways pay a "mileage credit" to shippers who provide their own equipment. This takes the form of a payment for every mile that a car moves over a particular railway. While the costs of leasing or ownership are largely "fixed" costs, "mileage credits" from the railways increase directly with utilization.

Consider the following example in which utilization is increased by 10 percent as a result of improved fleet management:

> A covered hopper car is leased at $550 per month and is used for shipments over a 3,000-mile route. Mileage compensation is paid at a rate of 12 cents per mile.

BEFORE		**AFTER**	
3,000 miles per month		3,300 miles per month	
Rental:	$550/mo.	Rental:	$550/mo.
Mileage Credit:	$360/mo.	Mileage Credit:	$396/mo.
Net Cost:	$190/mo.	Net Cost:	$154/mo.

For the shipper, this means:

Capacity is increased by 10%
Budget requirements are reduced by 19%
Costs per ton are reduced by 26%

This is an impressive return considering that a 10-percent improvement is easily achieved with a good information system and appropriate follow-up.

The Potash Corporation of Saskatchewan (PCS) is among North American shippers who have found it necessary to operate their own fleets of railcars. With a growing need for potash as a fertilizer ingredient, the Corporation has had no alternative but to operate its own fleet in order to maintain an acceptable standard of service to its customers. The fleet is now made up of 1,600 leased cars which are used to move potash over sixteen major railways to markets that are concentrated in the midwest states but extend to the eastern seaboard and the Gulf States. As the fleet grew, it became evident that close monitoring of the cars was essential and that this could only be accomplished on an "exception" basis. Consequently, a computer system was designed to isolate problems and to reorient staff effort from **finding** problems to **solving** them. The system was developed in stages over a 6-month period, with regular input from the ultimate users. The resulting system has features and flexibility that are "tailor made" to specific conditions and circumstances. Although the main function of the system is to locate problems that require action by railways, repair shops, or customers, the system also serves a number of "spin-off" functions.

System Design

The system is extremely simple in concept. It is based on two information sources — basic information on the fleet (and individual cars in the fleet); and an historic record of the use and movement of each car. Data is analyzed only when the Fleet Supervisor requires certain information and specific reports to meet his immediate requirements.

The fleet data includes a listing of car numbers, together with corresponding rental rates, rate of mileage compensation, type of car, etc. This information is used mainly for control purposes and for the selection and structuring of reports.

The history of car movements is created from Car Location Messages (CLM's) received from the railways on a daily basis. As shown in Figure 1, the CLM provides information on the location and status of an individual railcar at a particular point in time. This information is available "on demand" from the major railways and is provided in a standard format. Car status and location is updated by the railways on a regular basis, either by trackside scanners or by manual input to railway computer systems.

The CLM system is not as complete or as accurate as it might be. Many small railroads do not supply CLM information and consequently, cars "disappear" when they are moving on short-line railroads. In addition, about 2 percent of the records are inaccurate. These shortcomings produced a series of "false alarms", when the system was first introduced. Many "problems" were investigated, only to find that they reflected incomplete data or erroneous records. To the extent possible, these shortcomings are now taken into account in the system design and user procedures.

Each railway sends CLM messages by Telex every 24 hours, providing status reports for all cars in the PCS fleet, which happen to be on that particular railway, at that point in time. The information is captured from Telex by a data converter and stored on a cassette tape for entry into the computer system. A typical load for a 24-hour period would include about 1,000 CLM's covering about two-thirds of the fleet. (The remaining one-third of the fleet would not have moved since last reporting and consequently no CLM's are sent for these cars.) After tests for validity and consistency, some on-going status records are updated and the CLM's are added to the file of railcar movements for later analysis at the direction of the Fleet Supervisor.

One important check is made automatically as the data is entered — to make sure that no cars have been given to other shippers for loading. In this case, fast action will often be successful in returning the cars to PCS service before they are loaded by another shipper.

Fleet Performance

As already noted, the main purpose of the fleet management system is to keep cars moving — to move as much product as possible at the lowest possible cost. Efforts to improve performance are concentrated on cars which (apparently) are not moving. Any such cars are identified by searching the car status records for railcars which have not moved in a specified period of time, such as 48 hours. The standard (or threshhold) used by the Fleet Supervisor depends on a number of factors including weather conditions, railway performance and conditions in the potash market. Once identified, delays may require follow-up with: a) the railways, to correct general shortcomings in service or to move a particular car; b) repair shops, to deal with PCS cars on a priority basis; and c) customers, to unload a car and return it to the railways as quickly as possible.

To assist the Fleet Supervisor in isolating and investigating problems, the system allows the user to select and organize problems on the basis of a number of attributes. As shown in Figure 2, this enables the Supervisor to specify and concentrate on particular railways, groups of cars, or geographic territory. This is particularly useful for problems related to local weather conditions or "bottlenecks" on a specific railroad or railway yard. If necessary, the recent history of a particular car can be printed out in order to assist in understanding or dealing with a problem.

Experience has shown that the system is effective in isolating problems and expediting the movement of railcars. In addition, the railways are conscious that we are able to monitor their performance and consequently are less likely to favour other shippers at our expense. Flexibility has proven to be a valuable feature, particularly the ability to specify performance standards and tailor reports to conditions that prevail at any point in time.

Operations Planning

Information developed by the fleet management system is useful for operations planning, as well as improving railcar movement. The Fleet Supervisor must be able to anticipate how many cars will be available for loading at the minesites within the next few weeks. As an aid in making these forecasts, the report shown in Figure 3 provides information on the number of empty cars that are located at each minesite and on each major railway. With this information at hand and based on his experience with the railways, the Supervisor can anticipate the flow of equipment into the minesites through a period of two to three weeks into the future.

The system can also be used to investigate railcar movement at a particular location, in a specified geographic area, or on an entire railway. For example, if there is any indication of operating problems or labour problems on one railway or at a particular railway yard, special listings can be prepared in order to investigate or monitor those problems. If weather conditions are disrupting railway operations over a broad geographic area, developments in that area can be closely monitored with the system. Once it is clear that PCS traffic is being adversely affected and the condition is likely to persist, action can be taken to route traffic around that particular area. Staff can then use the system to make sure that cars are moved out of that area as quickly as possible and can monitor the performance of other routes that have been selected to avoid the problem.

Other Functions

The fleet management system performs a number of secondary functions. For the most part, these are simply extensions of the basic system, and make use of the same data sources as the basic system. These other functions include:

- **Time Lost for Repairs**: Some 2 to 3 percent of railcar capacity is lost while cars are out of service for repairs. The car status records are used to isolate any delays to individual railcars in repair shops. However, it it is also useful to monitor the repair requirements of the entire fleet or certain segments of the fleet. Figures 4 and 5 show examples of summary reports that are generated by the system. Figure 4 shows the amount of time that is lost for repairs and is a measure of the effective capacity of the fleet. Figure 5 shows the duration of repairs as an indication of the seriousness of repair problems. These reports are reviewed regularly for any sign of an unfavourable trend. They also serve as a measure of success in improving the condition and performance of the fleet.

- **Fleet Maintenance Records**: The fleet management system includes a record of all repairs performed on equipment. This includes details on the type and cost of repairs and where those repairs were carried out. Summary reports can be generated when required by the Fleet Supervisor, and can be used to monitor the performance of specific repair shops or the maintenance experience for a particular group of cars. This latter information is a major factor in deciding whether or not to renew leases. If a group of cars requires frequent and expensive repairs, it may be more economical to lease newer (more expensive) cars rather than continuing to pay heavy repair bills.

- **Delivery Times**: The fleet management system constructs a record of time-in-transit from Saskatchewan to selected geographic areas. With this information, it is possible to monitor delivery times from Saskatchewan to say, the State of Illinois over a given period of time in order to identify trends.

- **Customer Service**: Selected customers are given limited access to the system, allowing them to monitor the movement of cars to their plants. This allows them to estimate arrival times for their supplies and to take this into account in their operations planning.

- **Lost Cars**: As already noted, railcars are sometimes assigned to the wrong shipper. The system identifies cars which are placed for loading at any location other than PCS minesites. This enables PCS to notify the railway of its error and to arrange for the car to be removed from the shipper's siding and returned to a PCS plant for loading.

- **Assessment of Service**: The Fleet Management System can be used to monitor the movement of selected cars in order to evaluate the service on a particular route. This is accomplished simply by printing the accumulation of car movement records for selected cars once the routing has been established.

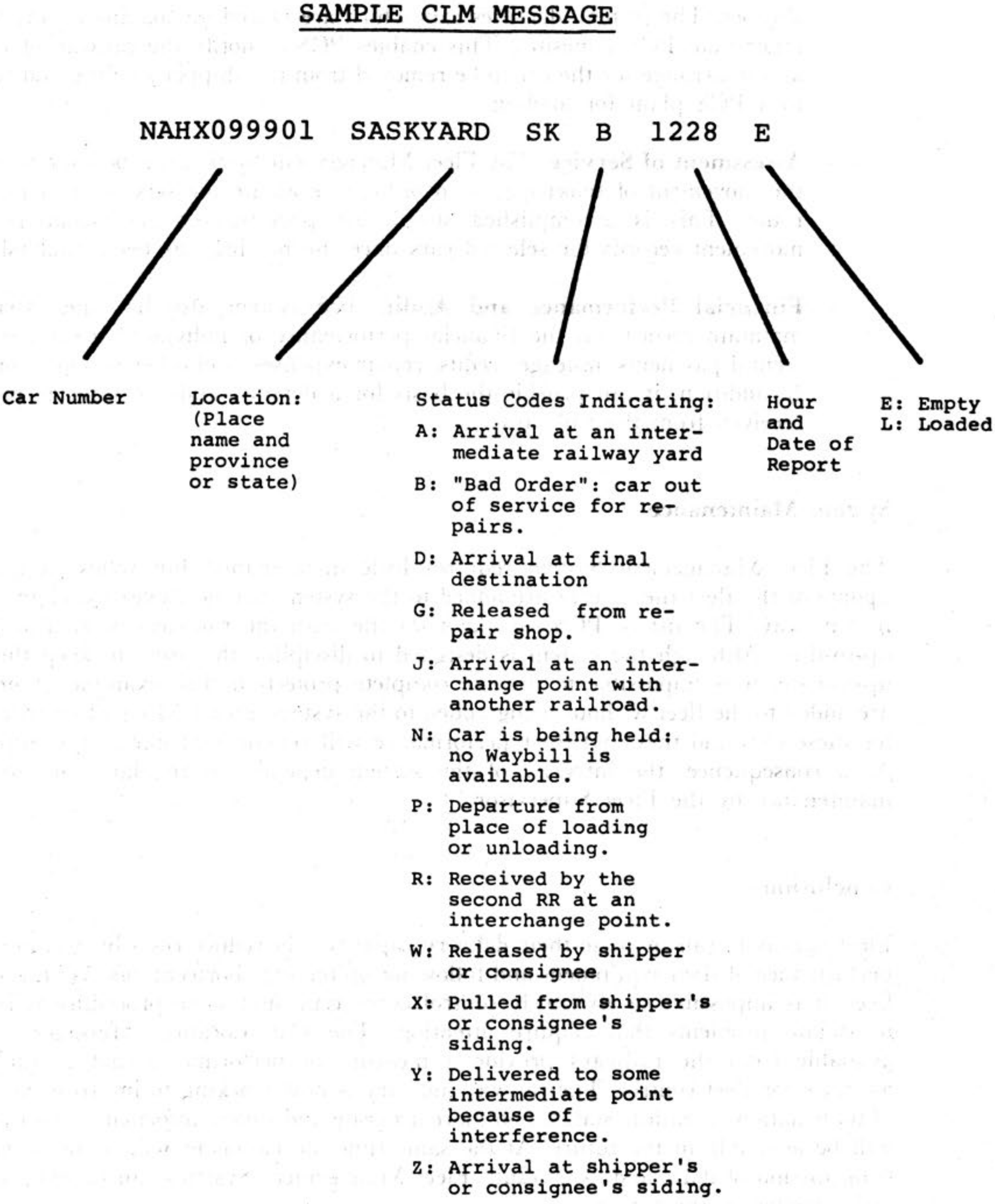
- **Financial Performance and Audit**: The system also has the capacity to maintain records on the financial performance of individual leasing contracts (rental payments, mileage credits, repair expenses, and other revenues or costs). In addition, it can provide the basis for a detailed audit of mileage payments received from the railways.

System Maintenance

The Fleet Management System requires little maintenance, but relies on a regular update of the fleet file as cars are added to the system, are destroyed, or change status in any way. The list of PCS cars kept on file with the railways must also be kept up-to-date. Although the system is designed to discipline the users to keep these files up-to-date, it is impossible to provide complete protection. For example, if new cars are added to the fleet without being added to the system, no CLM's will be transmitted for these cars and they and their performance will remain invisible to the Supervisor. As a consequence, the integrity of the system depends on regular and disciplined maintenance by the Fleet Supervisor.

Conclusion

Fleet operators can increase their delivery capacity and reduce costs by monitoring the performance of their equipment and following up on any shortcomings. With a sizeable fleet, it is impossible to effectively control every unit, and some procedure is required to isolate problems that require attention. The Car Location Messages that are available from the railways provide a measure of performance that is sufficiently accurate for fleet control. The railway industry is now working to improve the quality of information on railcar status, and more accurate and timely information on car status will be available in the future. At the same time, development will allow more rapid transmission of data, and as a result, Fleet Management Systems can be more efficient and effective in the future.



SAMPLE CLM MESSAGE

NAHX099901 SASKYARD SK B 1228 E

Car Number

Location: (Place name and province or state)

Status Codes Indicating:

A: Arrival at an intermediate railway yard

B: "Bad Order": car out of service for repairs.

D: Arrival at final destination

G: Released from repair shop.

J: Arrival at an interchange point with another railroad.

N: Car is being held: no Waybill is available.

P: Departure from place of loading or unloading.

R: Received by the second RR at an interchange point.

W: Released by shipper or consignee

X: Pulled from shipper's or consignee's siding.

Y: Delivered to some intermediate point because of interference.

Z: Arrival at shipper's or consignee's siding.

Hour and Date of Report

E: Empty
L: Loaded

Figure 1

OPTIONS FOR SELECTION OF PROBLEM AREAS

Variable	Attribute	Example
CAR	Car Number(s)	Car EQ NAHX 99901 THRU 99930
RR	Railroad(s)	RR EQ 'CN,SOO'
LOC	Location	LOC EQ 'SYLV,ALWI'
CODE	Refers to CLM status Codes shown in Fig.1	CODE EQ 'PJ'
STATE	State or Province	STATE EQ 'AL,DE'
DAYS	Threshold for delays warranting investigation	DAYS GT 5
RR DAYS	No. of days spent on one railroad	RRDAYS GT 20
TRIP DAYS	No. of days already spent on this trip	TRIPDAYS GT 50
PLANT	Minesite	PLANT EQ 'SYL,DUV'
CARTYPE	Type or railcar	CARTYPE EQ 1
CONTRACT	Lease identification	CONTRACT EQ 'NA0001, NA0002'
CUST	Customer Name	CUST EQ 'DSE'
STATUS	Car status (i.e. whether active, derailed, in repair shop, etc.)	STATUS EQ 7
LOAD	Loaded or empty cars	LOADED

Figure 2

DISTRIBUTION OF EMPTY CARS

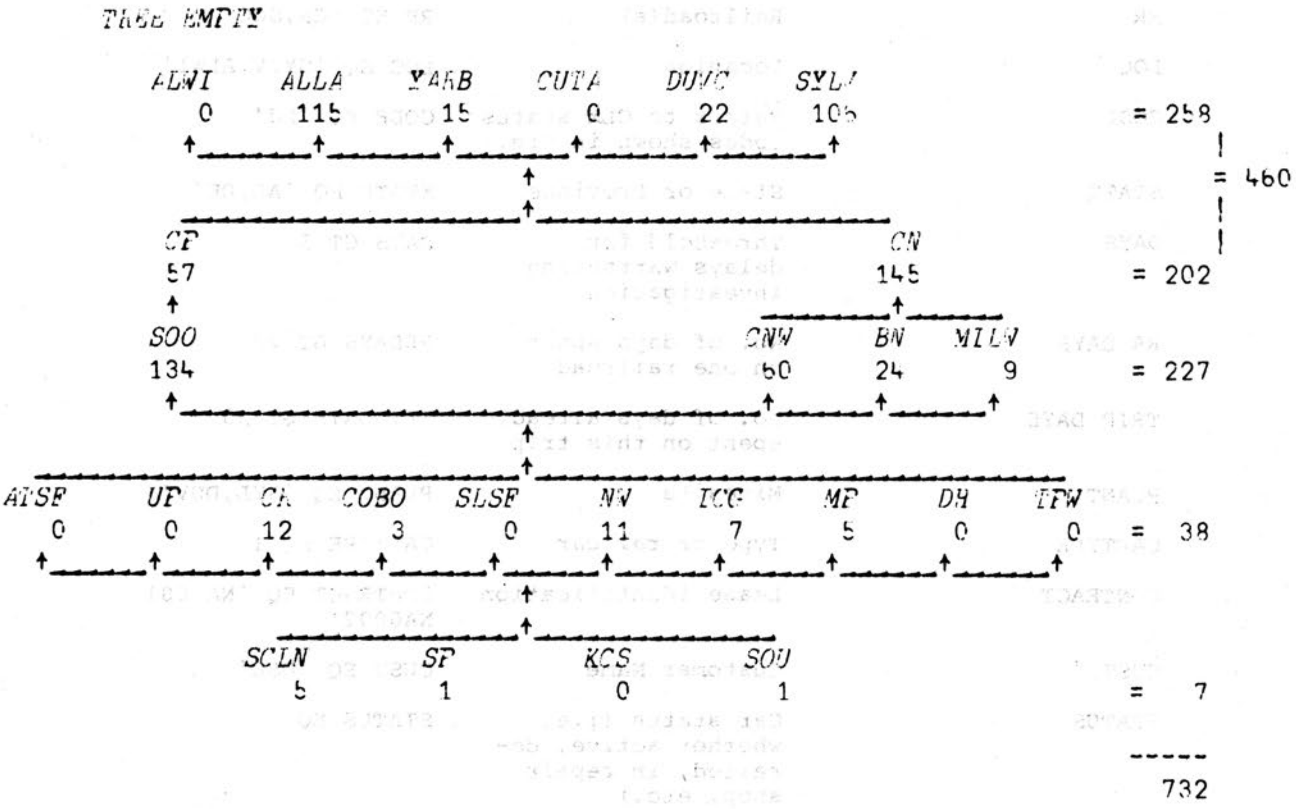


Figure 3

TIME LOST FOR REPAIRS

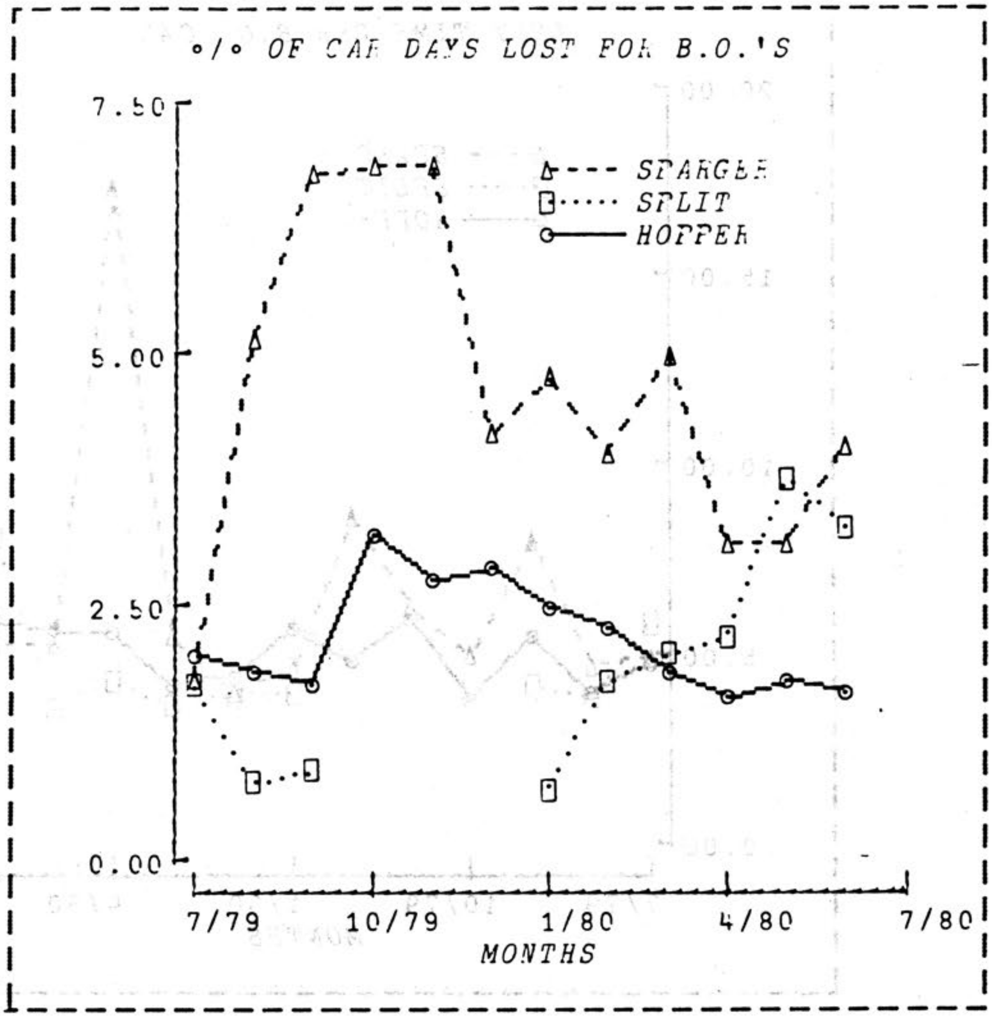


Figure 4

DURATION OF REPAIRS

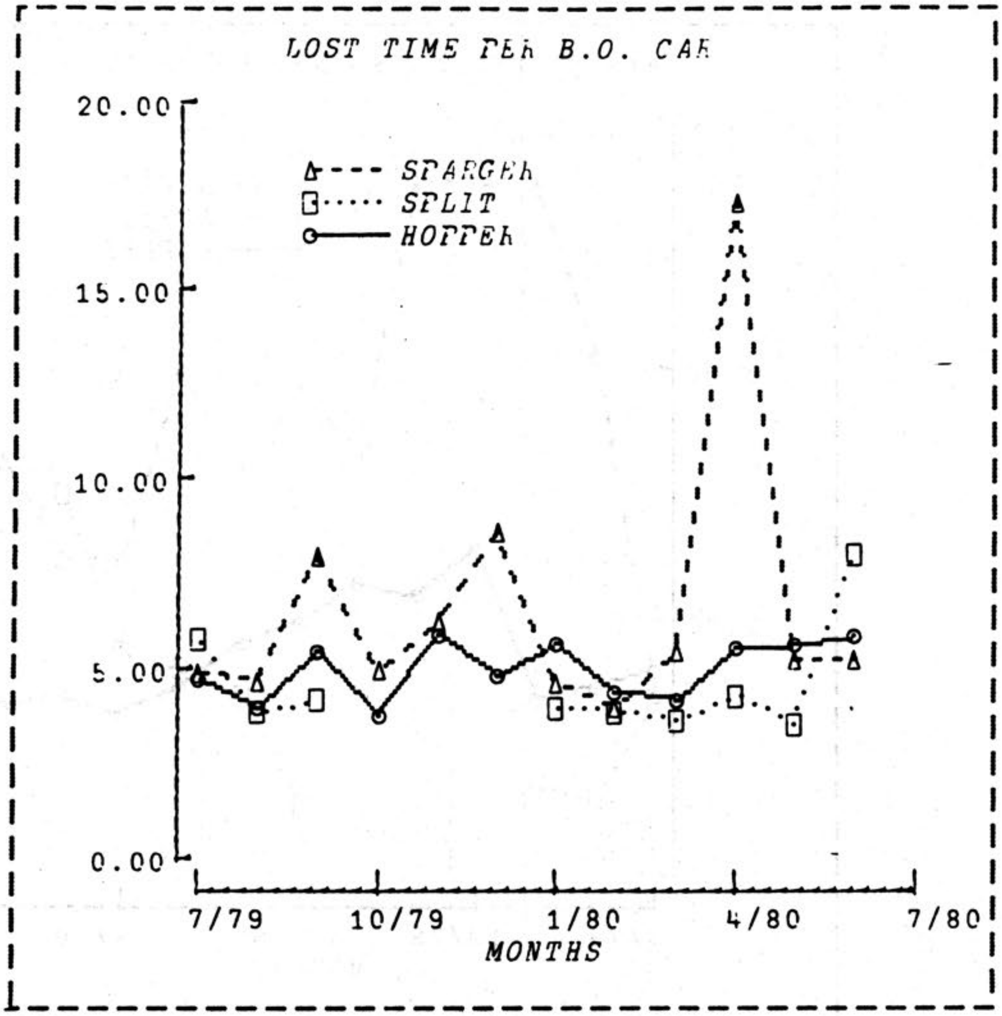


Figure 5

OPTIMATION OF APL CODE

Roy A. Sykes, Jr.
STSC, Inc.
Woodland Hills, California

Abstract

In a language as rich as APL there are frequently multiple solutions to a given problem. Optimization is the process of selecting that algorithm or design which minimizes resource consumption. More often, optimization constitutes substituting one primitive for another, various forms of pseudo compilation, reuse of identifiers, and tuning code to specific characteristics of data.

This paper discusses the characteristics of optimization in APL. Among the topics covered are: reasons for optimization, appropriateness, effect on productivity, how to optimize, and when not to optimize. Examples of optimized code are also provided.

Introduction

We all optimize. Our very presence here today is due to the efforts of a relatively small group of people in optimizing certain aspects of man-man and man-machine communication. Yet there is no question that we are far from our goal. Ask anyone — APL needs more structures, more datatypes, more functions, more operators, more space, more speed. And therein lies the paradox: no one knows what is optimum!

Certainly a program that can post the books and format an invoice for a tenth of a cent of CPU charges must be near-optimal. Yet if we're informed that the same program takes 12 megabytes of main memory, our opinion of it quickly drops to sub-optimal. The problem is that there are many simultaneous dimensions to optimize. Sometimes, pleasantly, they parallel, but more often they are orthogonal. Even two parallel dimensions can be reversed, so that optimizing one aspect diminishes the other — the common space-time tradeoff is an example.

In the context of this paper we will consider optimization as the process of making an APL program or system require minimal computer resources to solve a given problem. While such resources as static data storage, dynamic working storage, I/O, and connect time all contribute to an APL program's cost, the bulk of the cost typically derives from CPU time, and it is on this which we will concentrate. However, many of the points we will make bear equally upon other cost elements of a program. Note also that the techniques we mention to reduce CPU usage may have deleterious effects on some other aspects of our example programs. Nothing is free.

First, Decide What To Optimize

Amid the morass of tradeoffs and conflicting goals it is helpful to decide just what it is one wishes to optimize and at what expense to other considerations. For example, suppose we must interactively accept a long series of account numbers (five-digit integers). We expect the user will enter valid account numbers, but must assure ourselves by checking. The following simple program is adequate.

```
     ∇ R←GETACCTS ACCTS;A;B
[1]    R←⍳0 ◇ STOP←⋆1 ◇ 'TYPE STOP TO DO SO.'
[2]   TOP:A←,⎕ ◇ →(STOP∊A)⍴0
[3]    →(∧/B←A∊ACCTS)⍴ADD ◇ 'NOT FOUND:  ',⍕(~B)/A ◇ →TOP
[4]   ADD:R←R,A ◇ (⍕⍴R),' SO FAR' ◇ →TOP
     ∇
```

Unfortunately, if *ACCTS* has tens of thousands of elements, the repeated *A∊ACCTS* search on line [3] can be quite expensive.

One solution is to defer checking until all input has been accepted, thus searching for many elements at once, which decreases the per-element search time. This has disadvantages to the user who doesn't realize an error until it is many pages back in his input worksheet.

Another solution is to create a large incidence vector and, in conjunction with error-handling, subscript it:

```
     ∇ R←GETACCTS2 ACCTS;A;B;IV;⎕ELX
[1]    R←⍳0 ◇ STOP←⋆1 ◇ 'TYPE STOP TO DO SO.'
[2]    IV←99999⍴0 ◇ IV[ACCTS]←1
[3]    ⎕ELX←'→NF' ⍝ ERROR LATENT EXPRESSION
[4]   TOP:A←,⎕ ◇ →(STOP∊A)⍴0 ◇ →(∧/IV[A])⍴ADD
[5]   NF:B←A∊ACCTS ◇ 'NOT FOUND:  ',⍕(~B)/A ◇ →TOP
[6]   ADD:R←R,A ◇ (⍕⍴R),' SO FAR' ◇ →TOP
     ∇
```

For our "expected" user this should be much faster. Yet it has its own disadvantages: increased program complexity; higher initial set up time; greater storage requirements; and more costly error reporting. Is it worth it? One must decide on the basis of the resources available (people and machines), the expected costs (time and money), and the relative acceptability of various alternatives.

Thus, the process of optimization must always be preceded by analyzing the problem and the environment, and setting realistic goals. So-called "blind optimization" is usually one-dimensional, has no preset goals, and often disastrously ignores other valid concerns, of which the most frequently cited are readability and maintainability. Good optimization achieves its desired effect within a problem's constraints and without undue perturbation to other aspects.

Some Techniques of Optimization

CPU optimization is particularly attractive in APL because the gains for a given effort can be quite substantial compared to other languages — factors of 10, 100, or more are not uncommon. In fact, some APL programmers view optimization as a recreational activity, even when the program in question is already eminently suitable and cost-effective. Unfortunately, APL early developed a bad reputation from some optimizers whose bag of tricks consisted mainly of shoving the entire program up onto the first line:

```
     ∇ M←MOD X
[1]    M←((M≠1)×(⌈/M)=M←(1↓M)-¯1↓M←(X⍳X),1+⍴X)/X←X[⍋X←,X]
     ∇
```

The above pornography is an excellent example of how not to optimize. The penalty in readability is too great, and the resource gains are too small. A better version follows:

```
     ∇ R←MODE VEC;FREQ
[1]   ⍝ GIVEN A NUMERIC VECTOR [OTHER ARRAYS ARE RAVELLED] <VEC>,
[2]   ⍝ COMPUTES THE MODE (THE MOST FREQUENTLY OCCURRING NUMBER(S)).
[3]   ⍝ IF ALL NUMBERS OCCUR ONCE, THE RESULT IS ⍳0.  ORIGIN 1 ONLY.
[4]    VEC←,VEC
[5]    VEC←VEC[⍋VEC]
[6]    FREQ←VEC⍳VEC
[7]    FREQ←(1↓FREQ,1+⍴VEC)-FREQ
[8]    R←(FREQ=2⌈⌈/FREQ)/VEC
     ∇
```

For practically no increase in running cost, we have a program that a human being can decipher. The trivial change of *(M≠1)×(⌈/M)=M* to *FREQ=2⌈⌈/FREQ* recovers all the "savings" gained in *MOD*. As a rule, don't optimize trivially when the human cost increases dramatically.

The above example also illustrates a short-sighted optimization strategy. The programmer apparently felt that he already had the optimal algorithm and chose to apply certain mechanical transformations to reduce system overhead. Such transformations are sometimes called "atomic fixes" when done by hand. "compilation" (a misnomer) when done programmatically. Other examples of such "optimization" are removing labels and comments, diamondization, incorporating subroutines, replacing all *IF*'s or /'s in branch statements by ⍴'s, and so on. These techniques have questionable merit. They require programmer time to maintain and invoke the "compilation" process. They increase the complexity of code management and debugging. Furthermore, if the subject functions manipulate character objects intended for execution (as by ⍎ or *⎕FX*), automated compilation can be unreliable.

Yet with current interpreter implementations these techniques do have minor run-time benefits. Production APL functions can easily have more comments than code. Subroutines can be expensive by duplicating storage. If you feel compelled to use these techniques, keep the source code, and lock the object code. Nobody wants to look at a function with no comments, no labels, and hundreds of characters per line. The object code is for execution only; the source code is what you maintain.

Let's look at *MODE* again and take a longer view: perhaps our algorithm can be improved. Since we have the vector sorted, all we need do is compare each element against its predecessor to mark the end of each series of duplicate values. Computing the distance between the marks gives us the frequency of each value.

```
      ∇ R←MODEL VEC;FREQ;MARKS
 [1]   ⍝ GIVEN A NUMERIC VECTOR [OTHER ARRAYS ARE RAVELLED] <VEC>,
 [2]   ⍝ COMPUTES THE MODE (THE MOST FREQUENTLY OCCURRING NUMBER(S)).
 [3]   ⍝ IF ALL NUMBERS OCCUR ONCE, THE RESULT IS ⍳0.  ORIGIN 1 ONLY.
 [4]    VEC←,VEC
 [5]    VEC←VEC[⍋VEC]
 [6]    MARKS←(1↓(VEC≠¯1⌽VEC),1)/⍳⍴VEC
 [7]    FREQ←MARKS-¯1↓0,MARKS
 [8]    R←VEC[(FREQ=2⌈⌈/FREQ)/MARKS]
      ∇
```

The resulting function is only slightly more complex. Yet on large vectors it performs many times faster because the expensive dyadic iota is eliminated. The key to effective optimization is not in looking at the details of a problem, but in stepping back and rethinking the algorithm, or the problem itself.

Let's take another example (given in the January 1978 "Whizbang" in STSC **News**). We are given what appears to be a direct APL translation of a Fortran program:

```
     ∇ CF←RATE CASHFLOW DEP;I;N
[1]    CF←DEP ⍝ (⍴DEP)=1+⍴RATE
[2]    N←⍴RATE
[3]    →N↓0 ◇ I←1
[4]   L20:CF[I+1]←DEP[I+1]+CF[I]×RATE[I]+1
[5]    →L20×N≥I←I+1
     ∇
```

It runs successfully, but very slowly. A so-called "hotshot" goes to work on it and produces the following program:

```
     ∇ CF←R CASHFLO2 D;I;J;N;T
[1]    T←D[I←1] ◇ D←1↓CF←D
[2]    R←R+1
[3]    →N←((⍴R)⍴L̲),0
[4]   L̲:T←CF[J←I+1]←D[I]+T×R[I] ◇ →N[I←J]
     ∇
```

It's apparent that no substantive analysis has been done. The algorithm is identical. Some tricks have been employed to gain a few milliseconds here, a few microseconds there, but it's still warmed-over Fortran. True, the gains are impressive (about 30%), but what of those 10- and 100-to-1 speedups mentioned earlier?

A totally different approach is called for. A little analysis and experimentation can yield the following function:

```
     ∇ CF←RATE FASTFLOW DEP
[1]    CF←1,×\RATE+1 ◇ CF←CF×+\DEP÷CF
     ∇
```

which not only looks like APL, it runs like APL — roughly 80 times faster than the original solution. A good rule of thumb in APL is: don't gamble on the percentages — optimize for factors and orders of magnitude.

Special-case processing has unusual characteristics in APL, and is worth discussing briefly. Because most APL systems are interpretive, the detection of special cases often takes a significant amount of CPU time, sometimes more than the main algorithm itself. Therefore one must decide whether the savings from special cases justifies the detection code which is always executed, regardless of its effect. Rare cases should generally not be treated specially unless, by removing code from the general processing routine, it benefits all users. For example, if you must perturb your general-case algorithm to accommodate empty arrays, it may be beneficial to detect and process them separately. The primary benefit is not that empty arrays are processed quickly, but that the processing of non-empty arrays (the normal case) is expedited.

When Not To Optimize

There are many circumstances when optimization is inappropriate. The most important measure of any program is whether it works. Obviously, it doesn't pay to optimize faulty code! If it is difficult to determine whether a program works under all conditions, it is a likely candidate for later change. Thus, complex programs should not be optimized. Instead, break the program into more manageable parts, and only optimize those parts which are both stable and expensive.

The primary reason to optimize is to reduce running cost. For a program in an ongoing application, running cost is the cost of a single typical call to the function multiplied by the number of times it is invoked. If a function is called infrequently, it is unlikely to contribute substantially to the cost of the application — ergo, don't optimize. Similarly, if a function is called many hundreds of times, yet costs only a few cents to run, the potential savings are vanishingly small. Pareto's rule applies: 20% of your code (or less) will contribute 80% of the cost. Identify that crucial code before embarking on any cost-reduction campaign.

Inexperienced programmers should almost never attempt to optimize. Their primary goal is to understand an algorithm and translate it into APL. As experience builds, they will naturally become more aware of the costs, and of alternative solutions. If a beginning programmer has written an overly expensive but otherwise acceptable program, let an experienced programmer optimize it. Then praise the beginner for solving the problem.

Experienced programmers seldom optimize from the start. They realize it is more important initially to consider the architecture and algorithms of a system than to fuss about optimizing. As the system grows, it becomes (sometimes painfully) apparent where the major costs lie. Since the optimum was not an initial goal, the programmer often leaves himself much room for improvement.

This approach is not fatuous, it's realistic. Optimized code usually takes far more time to write. Changes (which are frequent in a new system) become much more difficult to make. Usage assumptions often are invalid. The code may never be exercised enough to recover the extra investment. Optimizing everything from the start not only wastes time, it can obscure the essential processes of a system. The programmer can get lost in a tangle of optimized code that obscures his view of the forest. Essentially, don't optimize until you can optimize the very process of optimizing.

Much APL code should never be optimized for CPU speed. Models of existing or proposed system behavior are for human elucidation. Code written for pedagogical or communication purposes must be as clear and straightforward as possible. The wider the audience, the more perspicuous your code should be. We do not mean to imply that you should not consider your algorithm carefully. Rather, once you have an effective statement of the solution, concentrate on ways of making it clearer, not faster. Choose mnemonic variable names, comment judiciously, group statements logically, branch rationally, and use subroutines appropriately. It may well be that there is a dramatically faster, but more opaque, solution. If so its place, if any, is in an appendix.

Rationale

There are some who would argue that APL code should never be optimized. They cite as justifications the declining costs of hardware, improvements in APL implementations, and the increasing costs of people. These trends are well-documented, but only marginally contribute to the question of optimization right now. If you are spending $10,000 per month in timesharing, or devoting most of a large computer to APL, and could well be spending half that, your bottom line is being impacted by today's realities, not tomorrow's promise.

Perhaps more important: as the capacity and capability of computers increase, so too do the size and scope of problems which they must address enlarge. A given piece of software may process a given amount of data for cost *C* today. It is commonly argued that since the cost of hardware will drop by *P* percent, that the cost of processing that data will drop to `C-C×P÷100` (optimizers: `C×P⊥ ¯0.01 1`). It is commonly neglected that the amount of data processed may well increase *P* percent or more. Thus, effective (i.e., not short-sighted) optimization of algorithms and file designs will pay long-term benefits. Furthermore, we are competing in a marketplace with less people-productive, but sometimes more machine-productive, languages. Our total costs are often measured against that of alternative solutions.

So clarity may have to be sacrificed to commercial reality; a 20% increase in programmer time can justify a 500% decrease in running cost. It is naive to think that sophisticated computers and APL implementations will somehow overcome simplistic designs and algorithms. Essentially, the decision whether to optimize or not, like any other business decision, must be based on expected return on investment. That return can often be substantial, but it must always be balanced against the human investment.

WHAT IS SYBRON AND DOES IT REALLY NEED IN-HOUSE APL?

Lynne C. Shaw
Sybron Corporation
Rochester, New York

Sybron Corporation is a diversified company with operations in twenty-one countries around the world. The company manufactures instrumentation, equipment, chemicals and supplies for the process industries, laboratories and professional health market, for water and waste treatment and other applications. The company's first United States operating units, established in Rochester, New York, were Pfaudler Company (1884), a manufacturer of glass-lined steel tanks used by breweries and other process industries, and Taylor Instrument Company (1851). Other units, in Europe, are older. Over the years, the company has grown through a complicated series of acquisitions and mergers.

A corporate management with a strong commitment to the local autonomy of each division has retained geographical and product diversity throughout this growth history. The name SYBRON was chosen in 1968 when it became clear (following the Ritter-Pfaudler/Taylor merger) that hyphenating the names of all our component parts no longer represented the desired corporate identity, particularly in the city where KODAK and XEROX started. This corporate identity has been promoted in the succeeding years to the extent that major divisions carry SYBRON as a part of their names. The resulting corporation, with sales of $724 million in 1979, is comprised of 123 operating units, focused in 32 profit centers within five operating groups. The company has offices, plants and laboratories in 21 countries, does business in twenty-five currencies, keeps books in twenty-two and consolidates all results monthly, in United States dollars.

In spite of Sybron's relatively loose corporate structure, as growth and diversity increased, corporate financial reporting requirements came to be a serious administrative burden at corporate headquarters from the standpoint of data-collection, consolidation, and analysis. Within the divisions, postal delays made coordinating a corporate reporting schedule with local financial management increasingly difficult. In 1975, a major goal set by the corporate finance department was to improve Sybron's corporate financial reporting mechanism. The availability of a telecommunications network provided by I.P. Sharp Associates offered access for all Sybron domestic divisions and several foreign ones. This network, combined with the power and flexibility of APL allowed us to begin the implementation of an automated financial reporting system. Now thirty-one of our thirty-two profit centers are linked to the corporate headquarters (and each other) through APL systems. Those systems, which began with an unsophisticated data collection mechanism to replace keypunched input to a COBOL system, have gained acceptance as they have expanded. APL systems are used extensively for the collection of information and the preparation and distribution of reports for financial data for control, operations analysis, and planning. A comprehensive personnel data base has allowed us to prepare EEO reports, verify compliance with numerous government requirements, and prepare and implement a participatory savings and thrift plan. Access centrally and at the divisions allows shared responsibility for data base updates and maintenance as well as shared access to report preparation facilities. Virtually every area of corporate management benefits from these systems, particularly since intercompany memos are most frequently distributed on-line. We have reached the point where users requesting computer services no longer ask "Why APL?", they ask "Why not?".

The answer to that question is economic. Annual APL time-sharing expenditures exceed a million dollars. APL, with quick development time and easy maintenance as very attractive features, offers a solution to the backlog problems common to more traditional data processing approaches. It is not, however, a panacea. The level of demand has consumed the resources available. Using commercial time-sharing services, many attractive applications are impractical because of cost factors. Although support from these vendors is generous, any development effort using outside consultants, whether provided free or under contract, requires time and attention from the Sybron corporate information systems staff. That staff, which has provided most development services up to this time, is reaching a point where maintenance and minor enhancements to existing systems are a full-time activity. Sybron must now decide whether to allow these constraints to limit APL activity or to change conditions so that controlled growth can continue.

The most obvious change available to Sybron is to move APL activity in-house. Reports from other users have assured us that such a move is possible and can be accomplished with minimum disruption of activities and little re-education of end users. Several vendors have offered to provide the necessary software and assistance. In its five years experience with SHARP APL, Sybron has, not without trauma, succeeded in establishing a high level of user confidence in the reliability of the on-line systems, so an easy transition is a vital factor in our decision.

The financial justification has been provided by a study of expenditures for all data processing within Sybron domestic divisions. The study, (using APL for data collection and analysis), was designed as an aid to long-range planning for corporate data processing. Detail was requested concerning the status quo of data processing hardware, software, personnel and outside services. Projections were also requested on planned acquisitions and development if no action was taken centrally. A third case was defined as the projection of costs if a corporate utility was to provide time-shared APL, support to those divisions without local data processing capabilities, and limited distributed processing. Cash-flow analysis and earnings per share impact were studied in depth. An estimated savings of approximately forty per cent in APL costs anticipated from a move in-house would provide impressive dollar savings as projected for the ensuing five years, although first year costs would be high. Any expenditures made centrally to reduce division outlays for service bureau assistance would add to this favorable picture.

General financial and technical assurances sufficed to move us ahead to the real planning stages of this project. As we proceed with detailed specification of hardware, software and personnel requirements, the magnitude of this decision becomes apparent. How much easier to be a customer!

Our hardware choices are limited. We cannot justify an Amdahl 470! As the choice narrows to an IBM 4341 dedicated to APL or a 3031 shared with one of our local divisions, we must carefully assess the impact of this decision on our users and (in the case of the 3031) on those of our host. Our tools are inadequate for one hundred per cent answers. We must accept scientific best guesses and much "gut feel". The 3031 is the preferred option because start-up costs are substantially reduced and implementation can be accomplished sooner. Because we have had only limited experience with hardware and computer operations at the corporate center, we will also be afforded the guidance of experienced staff. This solution will serve for two years if growth is carefully controlled. We cannot with certainty predict response time and and computer availability during peak periods and must postulate that there is enough flexibility in use patterns to resolve serious problems. Now 99.6 per cent "up-time" during scheduled hours becomes a very impressive statistic as we contemplate how to achieve it. We have become vitally concerned with backup and security provisions, heretofore so competently handled by our time-sharing vendors.

Software choices are limited for other reasons. Because of our concern for a transition as "transparent to the user" as possible, we have chosen to use SHARP APL. Our basic software choice is not difficult; however, many packages and libraries which we use casually have separate price tags. Utilities integral to essential systems are also priced separately and are expensive. Many others may not even be available since they are not released as product, but are "semi-public" or experimental. Still other riches are discovered almost daily in casual conversations with other users, leading us to wonder if it is possible to specify intelligently from such a variety. Since we anticipate converting, where practical, service bureau activity to the in-house utility, we must consider software from other vendors. We have the opportunity to evaluate impressive packages intending to make them available for general use. In so doing, we find they have impressive price tags as well, coupled with caveats about efficiency and compatibility on systems other than the vendor's. The goal of an integrated, powerful in-house system is not easily nor inexpensively realized.

Personnel requirements prove still another difficult area. Being understaffed has become a fact of life. We have recently been authorized to add to our staff in order to satisfy existing needs. Can we predict beyond that into our new environment what our real staffing requirements will be? Perhaps even more critical is the problem of finding enough qualified people. We accept the reality that we must orient anyone hired to the Sybron environment (estimated as a three to six month process). Can we afford to undertake programming training as well? Can we afford not to do so? The regular training programs of APL vendors and other in-house users are much more impressive now as we consider organizing the education process.

Support level, too, has been assumed and easily accepted from our commercial hosts. The SHARP APL Development Group will still be there; perhaps we will not need to build our own. Other support will now be our responsibility. With this will come the privilege of control and an opportunity to provide much-needed services. We see this opportunity as a continuing challenge.

In addition to providing a timely reactive service to user requests, the MIS group is in a position to effect significant savings to the corporation by anticipating the needs of the Sybron community. The programming staff should be able to develop (or acquire and modify) applications software to meet the overall needs of these users. Major savings may be accomplished by reducing amounts currently being spent with other time-sharing vendors. Some of these applications have been developed specifically at Sybron's request and represent a relatively minor conversion effort. Other (often very costly) applications involve proprietary software, but provide traditional solutions to common business problems in the financial and manufacturing areas. Taking the initiative in acquiring or developing these applications centrally and providing necessary interface for local or divisional use will allow Sybron to provide a service comparable to that offered by vendors who appear to have immediate solutions to each division's problems and sell these solutions separately and repeatedly. If the corporate staff is unable to respond to requests for systems development from division as well as corporate users, the dollar savings of in-house service will be diluted by consulting and other time-sharing fees as well as further unnecessary expansion of local hardware and personnel expense. Sybron divisions will notify us of our success by preferring in-house services to out-of-house alternatives.

Altogether, moving APL in-house is a step in a long-range plan for data processing in Sybron Corporation. The corporate staff will assume responsibility for service and support previously supplied by I.P. Sharp Associates and other service bureaus. As long as this support is outside, it can be viewed as limitless. We must undertake to maintain a satisfactory level of support with our own resources, which we know to be finite. Implementing this proposal means that corporate headquarters must strive to be a front runner in automation. The corporate staff must honor requests for direction, consulting service and support to divisions seeking assistance. As a central resource we must offer a practical alternative to current choices. The corporate utility must provide better solutions, distributed or centralized, and make available the resources to implement these solutions.

OPERATORS AND ENCLOSED ARRAYS

Bob Bernecky
and
Kenneth E. Iverson
I.P. Sharp Associates Limited
Toronto, Ontario

In this paper we propose the introduction into APL of three functions and three operators, present their definitions, indicate the broad classes of applications, treat their relationship with other proposals, and discuss the order in which they are being introduced into the implementation with which we work.

The two main functions are **enclose**, which produces a scalar encoding of its argument, and its left inverse **disclose**. The three operators concern application axes and forms of composition. They are important in their own right, but have special significance in conjunction with the enclose and disclose functions because they extend the applicability of all functions to the arrays produced by enclose.

The following conventions will be used throughout the paper:

- Direct definition of functions [1] will be used, as in *F* : +/ι*ω*.

- Expressions such as *G*←+.× will be used to assign a name to a derived function.

- Complementary indexing will be used in referring to axes. Thus ⎕*IO*-1 will refer to the last axis, ⎕*IO*-2 to the penultimate axis, and so on.

- 1-origin indexing will be used. Hence 0 will refer to the last axis of an array.

- Names beginning with the characters *F* or *G* will denote functions; all other names will denote variables.

A) Enclose and Disclose

The functions **enclose** (<) and **disclose** (>) are monadic functions defined as follows:

1. 0=ρρ<*A*

2. ><*A* is identical to *A*

3. If *A* is a **simple** array (i.e. of present APL), then >*A* is identical to *A*.

4. Catenation applies to <*A* as to other quantities of rank 0, and other structural and selection functions (such as reshape, compression, and indexing) are extended accordingly.

5. The function > (defined above only for an argument of rank 0) applies to each element of its argument *A* to produce an overall result of shape (⍴*A*),*S*, where *S* is the shape (necessarily common) of each of the individual disclosed results produced. For example, if *A*←(<⍳3),<3+⍳3, then ⍴>*A* is 2 3, and ,>*A* is ⍳6, whereas >(<⍳3),<⍳2 would yield an error.

The dyadic function **idem** (≡) yields a boolean scalar, which is 0 if the arguments differ in shape or in any one of their elements. Two elements *AI* and *BI* differ if both are simple and if *AI*≠*BI*, if one is simple and the other is not, or if ~(>*AI*)≡(>*BI*). In particular, ''≡⍳0 is 1.

The definitions of the functions ∊ and ⍳ are presently based upon comparisons for equality (=); their extension to enclosed arrays is based upon the function ≡.

B) Composition and Related Operators

The operators to be defined apply not only to functions, but also to variables, and the names of the operators are chosen to suggest their use in both contexts. For example, the operator ⍤ will be called **on** because the derived function *F*⍤*J* applies *F* **on** the subarrays specified by the axes *J*, and because the function *F*⍤*G* applies the function *F* **on** the result of *G*. All derived functions are assumed to be ambivalent, and the monadic and dyadic cases must both be defined.

1. The operator ⍤ (on). A variable argument to the operator ⍤ determines the **application** axes, that is, the axes of the subarrays on which the function argument applies. In the monadic case *F*⍤(<*IR*)*B*, the argument *B* is treated as a "**frame**" of shape *S*←(~(⍳⍴⍴*B*)∊*IR*)/⍴*B* which contains subarrays of shape (⍴*B*)[*IR*], and the result can be considered as a frame of shape *S* which contains subarrays each of the (necessarily common) shape *SIR*, which is the shape of the individual result obtained on applying *F* to one of the subarrays of *B*. The shape of the overall result is therefore *S*,*SIR*. For example, if ⍴*B* is 2 3 4 5, then the shape of ,⍤(<1 3) *B* is 3 5 8, the shape of <⍤(<1 3) *B* is 3 5, and the shape of ><⍤(<1 3) *B* is 3 5 2 4.

The dyadic case *A F*⍤((<*IL*),<*IR*) *B* is defined analogously, with *IL* and *IR* determining application axes for the left and right arguments, respectively. The shapes of the left and right frames must normally agree. However, if one is empty, then the corresponding argument is used with each subarray of the other argument. This provides, in effect, a useful generalization of the notion of scalar extension.

Scalar extension is also assumed for the argument of the operator itself; that is, in the dyadic case, *F*⍤*J* is equivalent to *F*⍤(2⍴*J*). Moreover, in the monadic case, the last element of *J* is used, that is, *F*⍤*J* is equivalent to *F*⍤(''⍴⌽*J*).

The subarrays of the derived function *F*⍤*J* are specified explicitly by *J*, and the frames are specified implicitly as the remaining axes. In the expression *J*⍤*F* the converse is true; *J* specifies the frames explicitly and the subarrays implicitly.

We will refer to the subarrays of a right (left) argument *B* to which a function *G* applies as the **right (left) *G*-arrays of *B***, and to the corresponding frame as the **right (left) *G*-frame of *B***. We will also refer to the left and right *G*-arrays as **cells**.

In what follows we will have occasion to specify identities which apply for a given argument only in the sense that they apply for each of the cells of the argument. We will indicate such a **cell identity**, by the symbol →←, rather than by the ↔ used for a full identity.

The composition *F*⍤*G* is defined by the following cell-identities:

```
    F⍤G B →← F G B
A   F⍤G B →← (G A) F G B
```

For example, if the shape of *B* is 2 3 4 5 and *G*←+/⍤(<1 3 4), then ,⍤*G B* is a 3 by 8 matrix, each row consisting of the ravelled sums over the last axis of subarrays of shape 2 4 5.

2. The operator ¨ (with). The derived function *F*¨*G* (read as *F* with *G*) is sometimes called the dual of *F* with respect to *G*. It differs from the function *F*⍤*G* only in that the function inverse to *G* is applied to each individual result. Formally,

```
  F¨G B →← GI F G B

A F¨G B →← GI (G A) F G B
```

In many cases of interest, the function *G* possesses only a right inverse or a left inverse, but not a true inverse; in such cases a formal inverse will be assumed to be defined for the purposes of the operator ¨. For example, the square and square root functions will be assumed to be formal inverses, as will enclose and disclose.

The same operator used with a variable argument *X* produces a monadic derived function by assigning *X* as one of the arguments of the function involved. Formally,

```
X¨F Y →← X F Y

F¨X Y →← Y F X
```

For example, ⋆¨2 and ⋆¨.5 are the square and square root functions, and the length function (that is, *L*:(+/⍵⋆2)⋆.5) could be written as ⋆¨.5⍤(+/⍤(⋆¨2)) or, since the square and square root are formal inverses, as +/¨(⋆¨2).

Again, if the left *F*-frame of *X* is not empty, each individual result of *X*¨*F* is the collection of results obtained by applying each left *F*-array of *X* as the left argument of *F*, and the shape of each individual result is therefore the shape of the left *F*-frame of *X* catenated with the shape of *X*1 *F Y*1, where *X*1 and *Y*1 are left and right *F*-arrays of *X* and *Y*. Thus, 1 2¨○ *A* will produce an overall result of shape (⍴*A*),2.

3. The operator ⍥ (along). Certain functions apply to an array by first splitting it into subarrays **along** some one axis and then treating the one-dimensional collection of sub-arrays produced. In present APL these include reduction, scan, grade, and permutations such as reverse and rotate.

The right argument of the operator ⍥ determines the axis along which the derived function applies. For example, if ⍴*A* is 4 5 4, the expression ⌽⍥2 *A* splits *A* into a collection of 5 subarrays along axis 2, reverses the order of the collection, and produces a result of the same shape as *A*; the expression +.×⍥2/*A* also splits *A* into 5 square 4 by 4 matrices, and produces the 4 by 4 inner product over the 5 matrices; +.×⍥2\A produces a collection of 5 inner products, giving an overall result of shape

4 4, 5; and +⍥0\A splits *A* (along the last axis) into 4 matrices of shape 4 5, and forms the four sums required by the scan into an array of shape 4 5, 4.

The placement of axes in the final result is determined as follows for the four cases of grade, permutation, reduction, and scan. No question arises in the cases of grade and reduction, since the former necessarily produces a single axis (i.e. a vector result), and the latter produces a single result of whatever shape results from the reduction. In the case of a permutation, the splitting axis retains its position so that the result always has the same shape as the argument, and in the case of scan the extra axis is placed after those produced by the several reductions which comprise the scan.

Applied to two **functions**, the operator ⍥ produces an alternate form of composition defined for the dyadic case as follows:

```
    A F⍥G B →← F A G B
```

The definitions of the various cases of the three operators ⍤, ¨, and ⍥ are collected in Table 1. A study of the table suggests two useful mnemonic devices: 1) the compositions *F⍤G* and *F⍥G* differ in the valence used for *G*, the larger valence being associated with the larger symbol, and 2) monadic cases are derived from the corresponding dyadic cases by removing all subexpressions involving the left argument.

Name	**Expression**	**Monadic**	**Dyadic**
On	*F⍤G*	*F G ω*	*(Gα) F G ω*
	F⍤I	Apply on axes *I*	
	I⍤F	Apply within frame *I*	
With	*F¨G*	*GI F G ω*	*GI (Gα) F G ω*
	X¨G	*X G ω*	
	G¨X	*ω G X*	
Along	*F⍥G*	*F G ω*	*F α G ω*
	F⍥I	Split along axis *I*	

Table of Operators

Table 1

4. Index origin. Although the value of *X* produced by the expression *X*←⍳5 depends upon the index origin in effect, subsequent changes in ⎕*IO* will not change the value of *X*. In the same way, the expressions *F⍤I* and *I⍤F* and *F⍥I* depend upon index origin, but the definition of the function *G* produced by an expression such as *G*←*F⍥I* is determined by the index origin in effect at the time of execution, and the behaviour of *G* is unaffected by later changes in ⎕*IO*. For example, if ⎕*IO*←1 and *G*←*F*⍥1 then the splitting axis of *G* is the **leading** axis, and remains so even though the **index** of the leading axis becomes 0 when ⎕*IO* is later assigned the value 0.

5. Rank of a function. The axes to which the derived function *F*⍤*J* apply are a property of that function, a property that must be known in order to apply the function properly. Although the application axes must be known for all functions, they have in the past been treated implicitly and have, perhaps, not received the attention they now merit.

For example, the fundamental definition of the matrix inverse function ⌹ requires application to two axes, and if it were formally defined to apply to the last two axes, then ⌹*A* would have the same meaning as ⌹⍤(<¯1 0), and the function ⌹ would extend usefully to arrays of rank greater than two.

Although the operator ⍤ allows the specification of an arbitrary set of application axes, it should, in defining the application axes of primitive functions, suffice to specify the number of axes to which they apply, and to assume that these are the final axes.

The number of axes to which a function applies will be called the **rank** of the function. Thus the derived function *F*⍤*J* has a left rank of ρ>''ρ*J*, and a right rank of ρ>''ρ⌽*J*.

Although ⌹ provides an example of a function that should probably be assigned a fixed rank of 2, it is commonly advantageous to assign unlimited rank to a function, since the operator ⍤ can always be applied to specialize it as required. For example, the inner product +.× has unlimited rank, and can be specialized in many useful ways, using expressions such as +.×⍤(2 1 ,⍤< 1 2), or +.×⍤ 3 1, or +.×⍤0.

C) Applications

Instead of attempting to provide a large number of examples of the application of enclosed arrays and composition operators, we will discuss two general classes of problems which they address, namely, representations, and the systematic extension of mixed functions to arrays of higher rank.

1. Representations. Although a number is represented by a sequence of digits and a word is represented by a sequence of characters, it is convenient to be able to treat each number and each word as a single entity. In APL, the former is possible, but the latter is not.

The type of "scalar" representation accorded numbers could be adopted to incorporate as scalars other things now representable only by non-scalar arrays: words, sentences, and paragraphs on the one hand, and complex numbers, natural numbers, polynomial coefficient vectors, and rotation matrices on the other. The adoption of some of these (such as complex numbers and words) might be practical, but the list is endless, and it is important to introduce a general method for producing and using scalar representations. The enclose function provides the ability to produce such representations, and the composition and other operators provide the facility for conveniently applying arbitrary functions to them.

For example, if *ENC*: (<(*K*-1)↑ω),*ENC*(*K*←ω⍳' ')↓ω : 0=ρω : ω, then *Z*←*ENC TEXT* is a vector of the enclosed words from the character string *TEXT*, and *Z*⍳<'*THE*' gives the index of the first occurrence of the word '*THE*'.

We will illustrate the matter of representation further by two numerical examples, the first concerning vectors of various shapes, and the second concerning arrays of various ranks.

The polynomial function PF: +/C×ω⋆ιρC can be represented by the vector C, and the representation of the product of two polynomial functions represented by C and D is then given by C P D, where:

```
P: α+.×((ιρα)∘.=(ιρ1↓α,ω)∘.-ιρω)+.×ω
```

A family of polynomials can then be represented as a vector V whose elements are enclosed vectors. and the representation of the product over such a family is then given by the expression P¨>/V.

For example, if V←(<A),(<B),(<C), then:

```
P¨>/V
(<A)P¨>(<B)P¨>(<C)              Def of reduction
(<A)P¨> <(><B) P (><C)          Def of dual
(<A)P¨> < B P C
<(><A)P >< B P C                Def of dual
<A P B P C
```

The Taylor's Series approximation to a function F can be represented by the values of the successive derivatives at some point X, and the approximate value of F X+S is then given as a sum of terms resulting from applying some dyadic function G to the argument S and to each of the derivatives. For the case of a scalar function F, each derivative is a scalar, and the function G is simply the product of (S⋆K)÷!K with the Kth derivative.

For the case of vector functions, the Kth derivative is represented by an array of rank K+1, and the Taylor's Series approximation at X can therefore be represented by a vector V of enclosed arrays of ranks 1, 2, 3, etc. The terms in the summation approximating F X+S are then (>V[1])÷!0, and (S+.×>V[2])÷!1, and (S+.×S+.×>V[3])÷!2, and so on. The approximation is therefore given by:

```
+/S¨G⍤>V
```

where G: α G α+.×ω÷¯1+ρρω : 1=ρρω : ω

2. Extension of mixed functions. Scalar functions, because they have rank 0 and produce results of rank 0, extend simply and systematically to arrays; mixed functions, which do not share these properties, do not extend so neatly. However, compositions with enclose and disclose provide a general extension scheme, because for an arbitrary function F, the related function F⍤>, has rank 0, the function <⍤F has a result of rank 0, and the function F¨> has 0 rank and 0 result rank, and therefore extends like a scalar function.

D) Related Proposals

The "strict" enclose function defined in this paper produces a result which differs from its argument in every case, and corresponds to the enclose function associated with "Axiom System 1" in papers on general arrays [2 3 4 5]. It therefore differs from the "permissive" enclose of Axiom System 0, which produces no change when applied to a simple array of rank 0.

The present proposal differs from most of the general array proposals in three major points: 1) the primitive scalar functions are not presently extended to enclosed arrays, and can be applied to them only through composition with the disclose function, 2) there is no definition of "fill elements" (exemplified in present APL by the space and the zero in the expansion and (over) take functions) for enclosed arrays such as those provided by the "prototypes" of the cited references, and 3) empty arrays are always simple.

However, as the concluding section makes clear, the present definitions leave largely open the possibility of such extensions, and the implementation plans will keep them open for some time to come. In particular, the use of strict enclose does **not** necessitate (as the permissive enclose does necessitate) the introduction of heterogeneous arrays [5], nor does it preclude their later introduction.

Readers familiar with other proposals may wish to make comparisons by using the functions and operators defined here to duplicate or approximate constructs in these other proposals. For example, the **each** operator [6] applied to a function *F* is equivalent to *F*¨> except for the case where every result produced by *F* is a simple scalar. The operators **each right** and **each left** can be approximated by expressions of the form *X*¨*F*¨>*Y*, and *F*¨*Y*¨>*X*. Moreover, the definition of a permissive enclose function *PE*: <ω : *SIMPLESCALAR* ω : ω would make *PE*⍤*F*⍤> equivalent to the **each** operator on *F* and, if the function *PD*: >ω were declared to be the formal inverse of *PE*, then *F*¨*PD* would also be equivalent to the **each** operator.

As a further example, the function **raise along axis** *I* defined by Gull and Jenkins [5], may be written as <⍤(<*I*), and is, in fact, more general then the raise function since *J* may specify a vector of axes rather than one. Finally, the function > itself (because it has rank 0) is a generalization of the Gull and Jenkins function **lower**.

As a final example, consider the indexing and other facilities proposed for applying functions at the "leaves" and other "levels" of an enclosed array. The recursively defined function:

F: *F*¨>ω : *T*ω : *G*ω

applies to its argument the function *G* at a level determined by a test of the items at each level performed by the function *T*. In particular, the function *F*: *F*¨> ω : *SIMPLESCALAR* ω : *G* ω applies *G* at the leaves.

The operators defined here are similar to those proposed in [7], although the symbols and names differ in most cases. In particular, the cases *F*⍤*I* and *I*⍤*F* were called **axis** operators, and the operator **with**, because it produces a derived function which is a **dual** of its left argument, was called the **dual** operator.

Gull and Jenkins [5] provide an extensive bibliography on enclosed arrays.

E) Related Functions and Operators

The three operators and three functions defined at the outset represent a rather minimal set required to introduce the use of enclosed arrays in an effective manner. There are, however, many related functions and operators which should receive some attention in assessing the utility of this minimal set. We will here discuss only one function and one operator in this class.

1. Format Function. The function is the extension of monadic format (⍕) to enclosed arrays, as proposed by Jenkins and Michel [8]. Briefly, the extended definition formats each enclosed element independently into a matrix, and pads them with spaces to form with their neighbouring elements a rectangular array of characters. There follows an APL model of the function and examples of its use. The examples were produced on SHARP APL, and in reading them one must recognize that the monadic format function on that system differs from some in not providing a leading blank column for all numeric matrices:

```
TH:(I,(⍴Y)÷(×/I←¯2↓J),1)⍴Y←1 C̲ C S⍉S((×/¯1↓J),¯1↑1,J←⍴ω)⍴E,ω
   :X≡>X←1↑,ω
   :⍕ω

  E : (<(⌽0,0⌊2-⍴⍴Z)↓H Z←TH>''⍴ω),E 1↓ω : 0=⍴ω : ''

  H:H(1⌽(¯2↓S),×/¯2↑S←¯1⌽⍴X)⍴X←(K+⌽2=+\1+0×K←⍴ω)↑ω
   :2≥⍴⍴ω
   :(¯2↑1 1,⍴ω)⍴ω

  S: ((MS A)S̲ A←(1↓⌽⍴ω)⍴⍉ω),S 0 1↓ω : 0=1↓⍴ω : ω

    MS: (¯1↑⍴>''⍴ω)⌈MS 1↓ω : 0=⍴ω : 0

    S̲: (<⍉((⍴⍴X)↑⌽2↑α,⍴X)↑X←>''⍴ω),αS̲ 1↓ω : 0=⍴ω : ''

  C: (<0 C̲ X),C 1 0↓ω : (×/⍴ω)=×/⍴X←(1↓⍴ω)⍴ω : <0 C̲ X

    C̲: X,[⎕IO+α×1<⍴⍴X←>''⍴ω]αC̲ 1↓ω : 1=⍴ω : >''⍴ω


      TH (<2 3⍴'ABCDEF'),<3 4⍴¯1+⍳12
ABC 0  1  2  3
DEF 4  5  6  7
    8  9 10 11

A:<2 3⍴'ABCDEF'
TA:<⍉>A
N:<3 2⍴¯1+⍳6
TN:<⍉>N
B:<'        '
BM:<0 8⍴''

      TH 3 2⍴B,B,A,TA,N,N

ABC    AD
DEF    BE
       CF
0 1    0 1
2 3    2 3
4 5    4 5
```

```
      TH 3 2ρBM,BM,A,TA,N,N
ABC     AD
DEF     BE
        CF
0 1     0 1
2 3     2 3
4 5     4 5

      TH 2 1ρTN,<A,TA
0 2 4
1 3 5
ABCAD
DEFBE
   CF

      TH Z← 2 2 1ρTN,<A,TA
0 2 4
1 3 5
ABCAD
DEFBE
   CF

0 2 4
1 3 5
ABCAD
DEFBE
   CF

      ρTH Z
2 5 5
```

If *X* is a single array, the expression *TH*<*X* can often be used to advantage. For example, if *X*←3 2 4 ρι24, then the shape of *TH*<*X* is 8 11, whereas the shape of *TH X* is 3 2 11, although the printed results are indistinguishable.

The utility of the extended function in report formatting can be seen from examples such as `TH X` and `TH 2 3ρ(3ρBM),X`, where:

`X←(<CITIES),(<POPULATION),<COUNTRY.`

2. Partitioning Operator. The convenience of enclosed arrays can be greatly enhanced by a "partitioning" function which partitions an array along some one axis according to some criterion (such as spaces or other delimiters in the argument, or an associated boolean argument), enclosing each of the parts, and catenating the enclosed parts. Smith [9] has discussed the use of such partitioning in the context of simple arrays.

Instead of a partitioning function we propose a partitioning **operator** of the form *F OP M*, where the two-row matrix *M* determines a partitioning of the argument of the derived function into a vector frame of subarrays to which the function *F* is applied individually. In particular, the derived function < *OP M* partitions the argument into a vector of enclosed subarrays, thus providing a partition **function** as a special case.

The partitioning occurs along some one axis, and the general case can therefore be easily obtained from the case of a vector argument of the derived function.

The non-negative integer matrix *M* determines the partitioning as follows. Each element of the first row *B* specifies the number of partitions beginning at the corresponding position, and the second row *E* determines the number ending at each position, in the following manner: if *I* is the first non-zero position of *B*, then `B[I]` partitions begin at position *I* and end at the first `B[I]` positions specified by `×E`, unless `B[I]>+/×E`, in which case the last `B[I]-+/×E` partitions continue to the end. The process is then repeated with `B[I]` set to zero, and with `1` subtracted from each position of *E* used.

The function *P* models the process, providing a boolean result whose successive columns represent the successive partitions:

```
P:(((⍴Q)↑E/0 1↓L,0)<Q←B/L←X∘.≥X←⍳⍴α),(α-B)Pω-E←ω⌊N≥+\×ω
 :0=N←+/B←α×<\×α
 :α∘.=''
```

For example, all contiguous partitions of order 4 are given by:

```
      4 3 2 1 P 1 2 3 4
1 1 1 1 0 0 0 0 0 0
0 1 1 1 1 1 1 0 0 0
0 0 1 1 0 1 1 1 1 0
0 0 0 1 0 0 1 0 1 1
```

The function *P* above employs the extended definition of compression called **replication** [11] which, for the vector case is defined by

```
REP: ((1↑α)⍴(1↑ω)),(1↓α)REP(1↓ω) : 0=⍴ω : ω
```

The following functions are convenient for experimentation:

```
ALPH: (R⌽B←' ',(K,1)⍴(1↓K←⍴R←⍉ω)↑'ABCDEFGHIJKLMN')[;;⎕IO]

TEST: (ALPH R),' ',⍉1 0⍕R←α P ω
```

```
      R←1 1 1 1 1 0 0 0

      R TEST ⌽R                    (R∨~R) TEST ⌽R                 R TEST R
ABCD       11110000          ABCD        11110000          A          10000000
 BCDE      01111000           BCDE       01111000           B         01000000
  CDEF     00111100            CDEF      00111100            C        00100000
   DEFG    00011110             DEFG     00011110             D       00010000
    EFGH   00001111              EFGH    00001111              E      00001000
                                  FGH    00000111
                                   GH    00000011
                                    H    00000001
```

The first two examples above show a "fixed window" and a "fixed window with trailing fragments". The arguments for other useful cases such as increasing and decreasing windows can be easily formed.

In the partitioning operator, the two vectors specifying beginning and end points are combined in a single 2-row matrix argument; it will probably be useful to also define the case of a vector argument in which a boolean vector specifies the beginning points of simple partitions.

F) Implementation Plans

Our implementation plans are guided by three considerations: to make the fundamental facilities available as early as possible, to defer further design decisions so as to benefit fully from experience in the use of the fundamental facilities, and to maintain complete compatibility for existing programs.

The essential functions have been in use by a limited set of users for some time, and are clearly of great practical value even without the composition operators. In particular, many applications would be simplified by using enclosed arrays instead of **packages** [10].

The functions in use include the comparison functions ∊ and ⍳ extended to enclosed arrays in terms of the **idem** function ≡, and all of the structural functions, including indexed assignment. However, because of the present exclusion of heterogeneous arrays, every array is limited to a single "type" (namely, numeric, character, or enclosed), and catenation and indexed assignment are therefore subject to certain domain errors which may subsequently disappear. Moreover, because no fill element has been defined for enclosed arrays, expansion or overtake of an enclosed array will produce an error.

Although the idem function does not distinguish between empty arrays of the same rank, expansion and overtake do. For the nonce, empty enclosed arrays are treated as numeric.

Implementation of new operators in a completely general way so as to apply to all functions (user-defined and derived as well as primitive) is a lengthy task, and in order to get some of the more essential facilities available for early experimentation we are beginning by implementing the following as special cases:

F¨*G* — The dual operator applied to an arbitrary primitive function *F* and a subset of the primitive functions *G*. The subset will include those monadic primitive functions of zero or unlimited rank which possess inverses, namely, > < ⋆ ⍟ + - ~ ⍉ and ÷. The case *F*¨> provides an inportant facility similar to the **each** operator. The very useful self-inverse functions ⌽ and ⊖ will be excluded at the outset because they raise questions of argument ranks other than zero.

F⍤*I* and *I*⍤*F* — The specification of the axes of application of a function is an important general tool. Specification of the frame is less important, but is easy to implement concurrently with *I*⍤*F*.

The present definitions extend to enclosed arrays only the structural functions and the mixed "comparison" functions (**idem**, **membership**, and **index of**), and nothing adopted either precludes or prescribes future extensions of other functions. In particular, the decision not to extend the equal function avoided any commitment concerning scalar primitives, a decision made practicable by the adoption of the related non-scalar function idem.

The new structures introduced into the APL workspaces to represent enclosed arrays were designed to address several problems:

a) They provide a uniform representation of all arrays, simplifying the implementation of the structural functions, and allowing them to perform equally well on all arrays, independent of the complexity of the underlying arrays.

b) They provide a general storage scheme which replaces multiple copies by more space-efficient multiple references, and permit more efficient comparisons in search algorithms. Moreover, the same structure may eventually be used to avoid the making of copies of arguments in effecting function calls.

References

[1] Iverson, K.E., **Elementary Analysis**, APL Press, 1976.

[2] Ghandour, Z., and J. Mezei, **Generalized Arrays, Operators, and Functions, IBM Journal of Research and Development**, July, 1973.

[3] Brown, J.A., **A Generalization of APL**, Ph.D. Thesis, Department of Systems and Information Sciences, University of Syracuse, 1971.

[4] More, T., Axioms and Theorems for a Theory of Arrays, **IBM Journal of Research and Development**, March, 1973.

[5] Gull, W.E., and M.A. Jenkins, Recursive Data Structures in APL, **Communications of the ACM**, Vol.22, No.2, Feb. 1979.

[6] More, T., **On the Composition of Array-Theoretic Operations**, IBM Cambridge Scientific Center, Report #320-2113, May, 1976.

[7] Iverson, K.E., **Operators and Functions**, IBM Research Division Report RC7091, 1978.

[8] Jenkins, M.A., and J. Michel, **Operators in an APL Including Nested Arrays**, Technical Report #78-60, Dept. of Computing and Information Science, Queen's University, Kingston, Ontario, March, 1978.

[9] Smith, Bob, **A Programming Technique for Non-Rectangular Data**, APL Quote Quad, Vol.9, No.4 — Part 1 (APL 79 Conference Proceedings), 1979.

[10] Berry, P.C., **Sharp APL Reference Manual**, I.P. Sharp Associates, 1979.

[11] Bernecky, Bob, **SHARP APL TECHNICAL NOTES**, I.P. Sharp Associates Limited, SATN 34, 26 AUG 1980.

Acknowledgements

We are indebted to a number of our colleagues for helpful discussions, particularly Doug Forkes and E.B. Iverson. P.C. Berry suggested the term "frame".

DOME SHARP APL - PLANNING AND ACHIEVING GROWTH

Ernst Goetze
Dome Petroleum Limited
Calgary, Alberta

Abstract

Dome Petroleum has used APL successfully in its financial planning, corporate economics, business development, marketing and engineering computer systems for a number of years. The author discusses Dome's requirements for growth in the use of APL, the decision to operate a sophisticated and comprehensive APL computing resource on its in-house computer system, the selection criteria and implementation of Dome SHARP APL, and the continuing programme of technical, human and corporate APL system resource development to achieve maximum growth and full potential in the use of this resource.

Introduction

Dome Petroleum Limited is a Canadian owned and controlled company, with an asset value in excess of $5 billion, engaged in the exploration and development of crude oil and natural gas. Dome's interests are primarily in Canada, including the Western Arctic where Dome is a leader in the development of Arctic drilling and oil and gas production and transportation technology. In addition, the Company has oil and gas interests in the United States and in the North Sea and operates a large natural gas liquids extraction, transportation, processing and wholesale marketing system in Canada and the United States.

Dome has used APL successfully for six or seven years at I.P. Sharp Associates and on an in-house Honeywell APL System to support key corporate computing in economics, financial and business planning, and engineering applications.

In March 1980, Dome became the first company to install and successfully operate SHARP APL under a current IBM MVS Operating System. This report describes our installation and is addressed to General Managers and Data Systems Managers who must now make similar decisions in order to meet the computing requirements of the 80's; APL, in over a decade of use and development, has achieved a recognized strength and importance in strategic corporate computing for which there is no other analogue in all of EDP today.

Installation - Test System

Exhibits I and II show the main events and requirements of our installation. As might be expected, we did encounter problems (Exhibit I) and there was still a great deal of work to be done (Exhibit II) before we would be satisfied with the operation and support of our in-house SHARP APL system in our IBM MVS environment.

The first system was installed for us as a test system by the I.P. Sharp Associates (IPSA) team, at no cost or obligation to Dome; a similar opportunity to install a test system of IBM VSPC/VS APL was given to IBM but was ultimately not accepted under the conditions that we offered (see Selection Criteria). Dome users were not involved at this point and we were careful not to make commitments that the system would be available this year, in order to assure the maximum objectivity of our Operations and Systems staff in evaluating the system for in-house operations.

IPSA sent a three-man team to Dome consisting of a Project Leader, System Programmer and Senior Operator. The IPSA team was confident that the system that they had developed for us in their VM/MVS test system in Toronto would be up and running in less than four hours. However, the MVS/SE system operated by Dome turned out to be significantly different from the MVS system and machine features that IPSA had in Toronto and it was three days before we had the first test run. IPSA then configured our libraries, established the system tasks and system files and spent the remainder of their time with us training our operators and documenting the system.

We ran the system in off-prime shift hours during the first five weeks of our installation in order to conserve computer memory during the prime time shift. We made a point, however, to run it every day even though there were no users, in order to familiarize our Operations staff with the essential procedures - startup, shutdown, full daily APL system backups, restores and the use of the operator and logging APL terminals that we introduced into the machine room. Secondly, we exercised all the functions of the APL processor, benchmarked the performance under simulated user loads of up to 60 concurrent users, evaluated the impact of SHARP APL on other commercial time sharing and vice versa; we installed the basic telecommunications support and we began the conversion of our Honeywell APL systems in order to prepare a familiar environment for selected users that we would allow on to the system when additional memory arrived at the end of April and we could operate during prime time.

The total system required 1.3 megabytes of page fixed main memory and one spindle of 3330-II DASD; we operated 4 incore swap slots of a moderate size, 170K-bytes, in order to promote superior response and reduce the memory requirement to an acceptable level.

Our benchmarks held no surprises and indicated that we could run up to 60 concurrent users with good response, though we would demand most of the available CPU of our IBM 3031 at that rate of use. We also noted that response (in user terms, the typical elapsed time of trivial tasks) degraded linearly in proportion to user load in contrast with the experience of several IBM sites that we had queried about the performance of VSPC/VS APL; at levels of use of VSPC/VS APL of 5-10 concurrent users executing tasks similar to ours, VS APL would have required a working set in excess of 1.5 megabytes and a high channel utilization (about 30%) and high CPU utilization which seemed closely connected with the high paging rates required by the processor, that would have discouraged further growth in our APL use on the existing resource.

The valuable results of our test installation were:

(1) The IPSA staff learned the requirements of our commercial general purpose data processing environment, the operation of the MVS master console, TSO/SPF, and our machine room practices.

(2) IPSA gained experience with MVS/SE and discussions with our staff have led to simplifications and necessary changes in the APL/MVS interface at each of two succeeding visits to our site in July and August.

(3) Increased respect among our Systems and Operations staff for the dedication and abilities of the IPSA team.

(4) Confidence among our Systems and Operations staff that the SHARP APL processor is operable and maintainable under MVS and that SHARP APL would make a valuable contribution to computing to Dome.

We were satisfied with the performance and reliability of our SHARP APL system by the end of the test period but still required:

(1) Utility Support, Operator Documentation and additional Operator Training (May 5-16)

(2) Attached Processor Support (July 1-8)

(3) IBM compatible VTAM Support (August 29-Septmber 5)

(4) System Documentation (August 29).

Attached Processor Support

IPSA believed, from a study of the IBM system documentation, that MVS could be forced to schedule and execute APL tasks on the host processor only; that turned out not to be true and the resolution of the problem required five days of testing and software development at our site, as well as further system changes applied in the August release to correct a reliability problem that became apparent during July.

Also by early July, our APL development programme (Exhibit IV) was well under way and we could no longer afford that APL not be available during prime time, so we ran the IBM 3031-AP in uniprocessor mode until the APL Attached Processor support was available. The dual support - uniprocessor and Attached Processor - has proved valuable since then since the Attached Processor has failed to operate on several occasions. The development of the Attached support has led IPSA to further simplifications of the APL/MVS interface and, in the August release, most of the system code previously retained in the system nucleus was moved to the link pack area in order to simplify host system maintenance and the installation of new APL releases.

VTAM Support

Dome supports a dedicated IBM 3705 running the IPSA Network Software for APL telecommunications support. However, Dome requested and IPSA developed a Shared Variable and Auxiliary Processor interface to IBM VTAM Network Support. The VTAM Support was installed at our site at the end of August and enabled all of our 3270-type devices for APL access. The initial support consisted of APL terminal access through 3270-type screens, multiple screen paging, programmable function keys and the first release of the IPSA Full Screen Management facility.

Dome SHARP APL Network Support

We have emphasized Network Support (Exhibit III) and currently offer:

(1) Worldwide access to our in-house SHARP APL system through the IPSA Network.

(2) Local direct line access at high speed (1200 baud) through a dedicated IBM 3705 and ALPHA concentrator running the IPSA Network Software.

(3) Bisynchronous Telecommunications Support in order to support local printing devices such as the IBM 6670 Laser Printer/Copier in user areas.

(4) VTAM support in order to enable 3270-type devices for user APL applications as well as other commercial time sharing at our site.

A particularly useful feature of the SHARP Network direct line (or dial-up) support is that our users can access both Dome SHARP APL and IPSA SHARP APL through the same line, at high-speed line rates, by a simple variation of the sign-on procedure.

In addition, by reducing the requirement for dial-up access to off-site access only, we have reduced our potential security exposure. Our network costs are low, compared to the cost of increased use of the SHARP Network at the rates of use that we anticipate.

Conversion and Migration Programme

It was very important to the success of our programme that the SHARP APL system would be available to us during the April-May-June-July window and that we converted key corporate applications from the Dome Honeywell system as quickly as possible; after all, conversion and migration of existing programs is not productive for the corporate purpose and we were not developing new Honeywell applications either. We began the conversion in late March; our first milestone was to convert enough of our Financial Planning applications by early May that selected users could access these, gain experience and a good opinion of Dome SHARP APL and continue development work for our Budget cycle processing, which began in July. If we missed that window and did not gain the confidence of this group of Dome users, we would not have succeeded this year since the Budget cycle and Forecast cycle extend into year-end and cannot be disrupted; the users would have had no choice but to use the existing programs. We hired additional staff at the beginning of May; two-thirds of our staff was then dedicated to new system development on the Dome SHARP APL system, while we continued on with the conversion programme. By early July, the user demand to convert from the Honeywell system or to migrate existing applications from the I.P. Sharp system had picked up dramatically; by the end of August, all of our Financial Planning Section, most of Business Development and several of the economists were using the system daily. It remained to migrate the Dome GOFER (Gas and Oil Field Economic Reporting) application to in-house; approximately 30-40 engineers and economists would then be converted simultaneously.

The load would have stressed our resources - staff, hardware, network - beyond what we could have handled in early September; we therefore introduced pilot production in September among selected users and targeted for full production in early October. We used September to assure the network support, upgrade to IBM 3350 DASD support (end of August), to make presentations to users and also to stabilize the operation of Dome GOFER in-house.

System and Application Program Support

We minimized the migration problem of separating Dome code from IPSA code in our IPSA applications by acquiring most of the I.P. Sharp Application libraries used by Dome. In addition to providing valuable software for our Dome libraries, we were also able to provide Dome users with a familiar environment. We have not made use of the IPSA System Development and Maintenance tools (APE, PAPER, etc.) nor of the IPSA APL Accounting System beyond the creation and maintenance of the Dome accounts; these applications have not been learned by Dome MVS Systems and Operations staff, nor has there been any requirement to; standard MVS procedures complemented by APL operator aids executed from the operator terminal are all that are necessary to operate and maintain our in-house system. However, future releases of APE that include a reduction of the library contents to those specific to our installation and specific APE-oriented system installation and maintenance documents are likely to increase the value of APE as an APL system maintenance tool.

Selection Criteria

Dome users have had 6-7 years of experience with APL use on IPSA SHARP APL and on the in-house Dome Honeywell APL. The SHARP APL service was adequate but did not offer us good potential for rapid growth in our use of APL (Exhibit V); our Honeywell APL Service was unreliable and scheduled to be phased out in 1982. The first of our IBM mainframes arrived at the end of 1979 and we planned to offer an APL service on the IBM equipment. IBM and I.P. Sharp Associates made presentations to our Data Systems Division in October, 1979; I.P. Sharp had not yet had a successful installation of SHARP APL under MVS and the product seemed very expensive. On the other hand, our inquiries among users of IBM VSPC/VS APL indicated that the competing IBM product had performance and reliability characteristics that would not be acceptable to Dome; we also believed that user acceptance of the product would be low and that the limited functionality of the product, compared to SHARP APL and Honeywell APL, would seriously impair our conversion, migration and system development programmes. Data Systems therefore insisted that we have the opportunity to operate both processors in our environment, without cost or obligation, in order to resolve these conflicting claims. I.P. Sharp Associates readily agreed to our conditions of the test and sent their installation team in early March 1980; IBM did not accept the conditions of the test and were therefore not considered any further at that time. However, the fact that there was only one candidate in the field did not change our standards; we went to Toronto in April 1980 in order to confirm our standards (Exhibit II) with I.P. Sharp Associates once we were convinced that the product was viable. By August, IPSA had earned our contract, both in the opinion of Dome users and of the Data Systems Division.

User Requirements

User requirements were established by our knowledge of the current applications, interviews with Dome users and senior management, the requirements of Dome Data Systems (Exhibit II) and the requirement for rapid and long term growth in our use of APL.

Summary of General Concerns

1. Reliability, performance, security of the APL system.

2. In-house consulting and system development support.

2. User steering and prioritization of the APL projects.

4. Efficient transition to the in-house system without impeding current applications.

5. Network support, particularly, high-speed direct lines, VTAM support, screen support, graphics support, etc.

Specific Concerns of Senior Management

1. Financial and Business Planning and Economics consulting support and system development.

2. Productivity gains through useful computer systems that support key corporate staff.

3. Acceptance of the in-house APL system by Dome users.

4. Reduction of the average X-weeks lag between the geologist's report, calculation of the cost to drill, calculation of the production economics and the economics review to less than Y-week(s).

5. Emphasis on computer systems that show us how to make money rather than how we spend it.

6. Engineering support, improved tracking of Dome wells, calculation of reserves and production economics.

We developed the Selection criteria (Exhibit VI) on the basis of these requirements; we compared

(A) IBM VSPC/VS APL vs. increased use of IPSA SHARP APL.

(B) Increased use of IPSA SHARP APL vs. DOME SHARP APL.

Our assessment (Exhibit VI) should be clear from the preceding remarks.

Dome SHARP APL Economics

The key to the economics of Dome SHARP APL is to anticipate growth in the use of APL; IBM VSPC/VS APL is more economic at low levels of use; IPSA SHARP APL provides a satisfactory service to some threshold of economic pain which is typically not more than 10 concurrent users logged on most of the day; beyond that, one should look to providing an in-house APL service, provided one can also provide the features and reliability that the I.P. Sharp APL service is noted for.

We anticipate growth in terms of the number of projected users (Exhibit VII), peak day concurrent loads (Exhibit VIII) and increases in user productivity, as measured by CPU demand per user per computer connect hour (Exhibit IX); Summary figures are shown in (Exhibit X) and some supporting detail is shown in (Exhibit XI). We have forecast growth and the economics of growth through to 1985; we expect, however, that our commitment to Dome SHARP APL will be much longer and that our returns on the Dome APL System development will be much higher than the mere cost of our investment in the system and the cost of support staff.

Key Assumptions

1. We anticipate that the **number of potential users** will grow rapidly initially but then tend to slow down by 1983; we feel that that is consistent with the kind of computing done by the APL applications. The corporate requirement for strategic planning and analysis does not grow in the same way as other data processing requirements.

2. The **connect time** per user would stay fairly constant since the network and system technology available to the in-house user would improve the productivity of his computer sessions; at the same time, there would be more reasons to access the computer because of new system development.

3. The **maximum number of concurrent users** during peak times is a relatively high proportion of the total number of potential users; this is to be expected if the computer is providing valuable results in our planning activities.

4. Our **average costs per connect hour** to the I.P. Sharp APL system are between $60-$120/hour, tending to $100/hour if we promoted increased use of IPSA SHARP APL by Dome staff; prices were escalated at 3% per quarter, but the discount offered to large users was increased from 30% in 1980 to 50% in 1985.

5. The **cost of operation** of Dome SHARP APL includes IPSA contract charges, the $100,000 cost of installation, Dome network operation, computer hardware and terminal costs.

6. The **APL support staff** begins with 3 in 1980 and grows to only 7 through 1981-1985; the cost of support staff is the essential cost of software development.

7. **Other non-specific costs** are estimated as 6% of the above, per year.

The **potential savings** (Exhibit X) are based on the difference between the anticipated cost of continuing to use I.P. Sharp's APL service and the use of Dome SHARP APL on our in-house computer. The **opportunity cost** of failing to develop a strong in-house support of APL computing, in terms of our ability to operate the resource, our staff support and the new systems development for key corporate strategic computing is not calculated; we expect to succeed and our early successes, aided by the excellent operation of our in-house APL system, shows us every promise that we will succeed. We expect that the value of that, over five years, will be incalculable compared to our failure to do so.

EXHBIT I DOME SHARP APL INSTALLATION

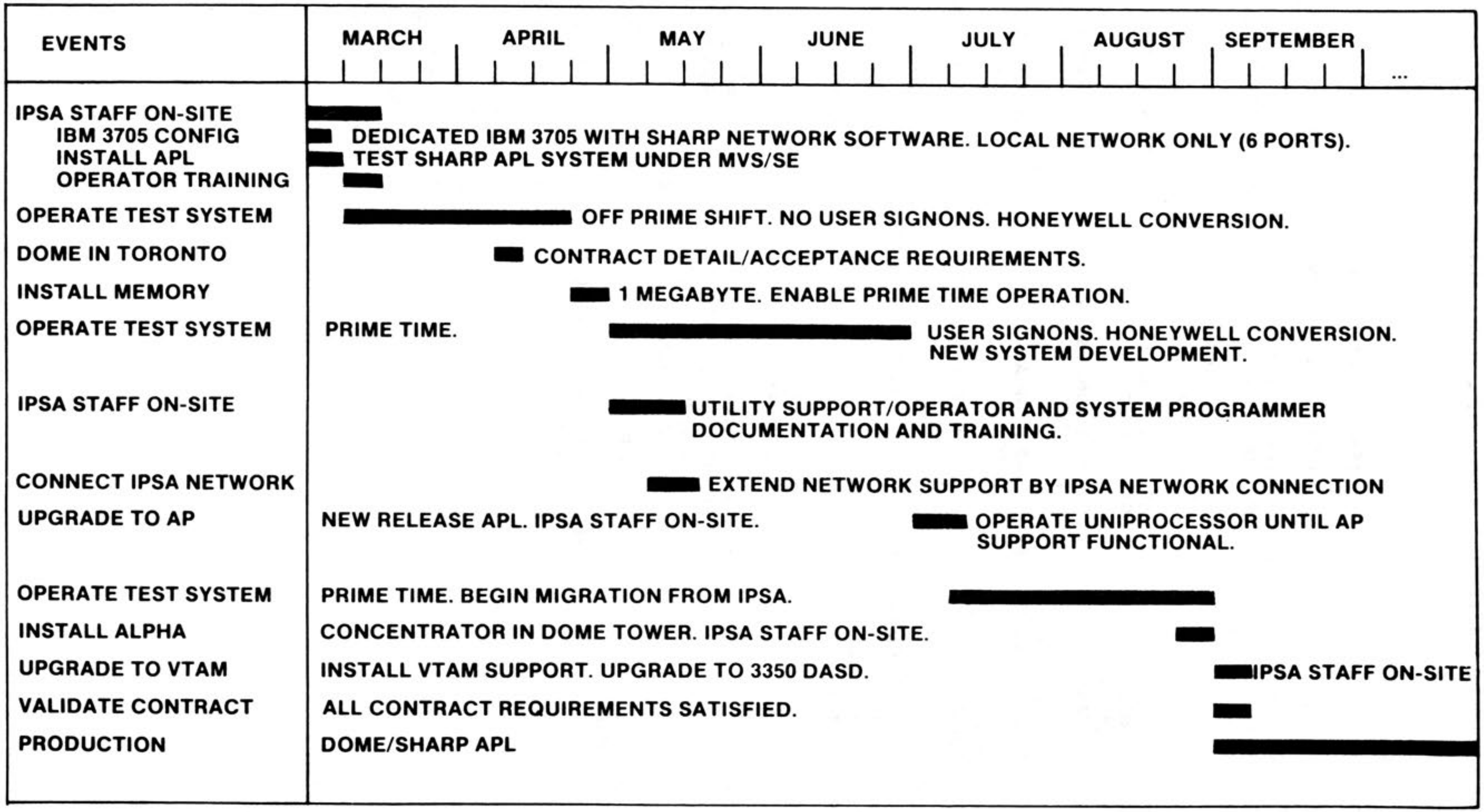


EXHIBIT II DOME SHARP APL INSTALLATION REQUIREMENTS

SYSTEM

(1) WS UTILITY - APL
DOWN
DUMP-DOME FORMAT
-IPSA FORMAT
RESTORE-DOME FORMAT
-IPSA FORMAT

(2) BSUT-APL DOWN
FDUMP-DOME FORMAT
-IPSA FORMAT
FRESTORE-DOME FORMAT
-IPSA FORMAT

(3) WS UTILITY-APL UP
SELDUMP-DOME FORMAT
-IPSA FORMAT
RETRIEVE-DOME FORMAT
-IPSA FORMAT

(4) FUTL-APL UP
FSELDUMP-DUMP FORMAT
-IPSA FORMAT
FRETRV-SOME FORMAT
-IPSA FORMAT

(5) BSSHOVE TAPE TO FILE
DSN TO FILE
FILE TO TAPE
FILE TO DSN

(6) RJE/RJO HOST

(7) HSPRINT

(8) DOME 3605-IPSA
CALGARY

(9) VTAM SUPPORT

(10) AP SUPPORT

(11) 3350 DASD SUPPORT
(12) DOME ALPHA
BISYNCH SUPPORT
(13) MIGRATION-IPSA USERS
(14) SYSTEM TUNING
(15) MAINTENANCE
WS ENTENTS
FILE EXTENTS
(16) SYSTEM TASKS
(17) BACKUP TO IPSA
(18) SECURITY PLANNING

SYSTEM DOCUMENTATION

(1) FUNCTIONAL OVERVIEW

(2) SYSTEM CONTROL
BLOCKS/TABLES

(3) APL SUPERVISOR-
STRUCTURE

(4) APL INTERPRETER-
STRUCTURE

(5) APL FILE SUBSYSTEM-
STRUCTURE

(6) SHARED VARIABLE
INTERFACE

(7) AUXILIARY PROCESSOR
SUPPORT

(8) RESOURCE REQUIREMENTS
MEMORY
DASD
CPU

(9) TUNING-APL
-APL IN HOST

(10) TELECOMMUNICATIONS
SHARP 3705 SOFTWARE
SHARP ALPHA SOFTWARE
LINE PROTOCALS

(11) UTILITIES

(12) INSTALLATION PROCEDURES
UPDATES
FULL

OPERATOR DOCUMENTATION

(1) OPRFNS, NUMBREQ
STATE, MSTEW

(2) SYSTEM COMMANDS

(3) START UP/SHUTDOWN

(4) BACKUP-APL DOWN
DUMP-DOME FORMAT
-IPSA FORMAT
RESTORE-DOME FORMAT
-IPSA FORMAT
FDUMP-DOME FORMAT
-IPSA FORMAT
FRESTORE-DOME FORMAT
-IPSA

(5) SYSTEM ERRORS

(6) COREDUMPS

(7) CRASH RECOVERY

(8) BSSHOVE

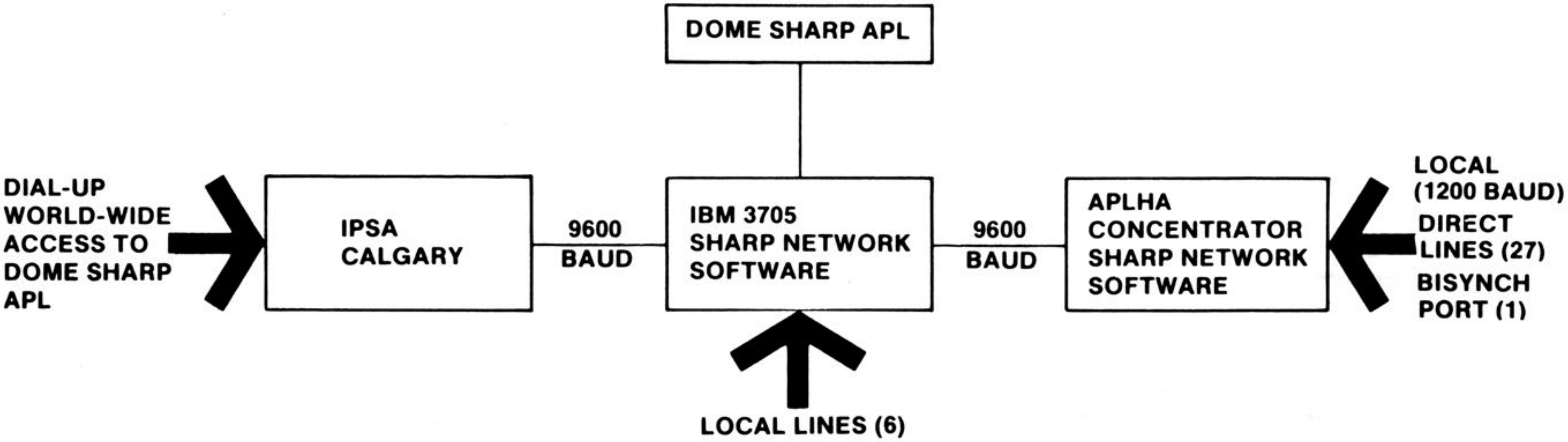
(9) WS UTILITY-APL UP
SELDUMP-DOME FORMAT
-IPSA FORMAT
RETRIEVE-DOME FORMAT
-IPSA FORMAT

(10) FUTL-APL UP
FSELDUMP-DOME FORMAT
-IPSA FORMAT
FRETRV-DOME FORMAT
-IPSA FORMAT

(11) NEW ACCOUNT
BFMTN
NUMBREQ
MSTEW
(12) DOME BILLING
(13) IPSA BILLING
NETWORK USE CHARGES
(14) SECURITY PLANNING
(15) BACKUP TO IPSA
(16) TP OPERATION

(17) HSPRINT

EXHIBIT III DOME SHARP APL NETWORK


DOME SHARP APL
DIAL-UP WORLD-WIDE ACCESS TO DOME SHARP APL
IPSA CALGARY
9600 BAUD
IBM 3705 SHARP NETWORK SOFTWARE
9600 BAUD
APLHA CONCENTRATOR SHARP NETWORK SOFTWARE
LOCAL (1200 BAUD)
DIRECT LINES (27)
BISYNCH PORT (1)
LOCAL LINES (6)

DOME SHARP APL

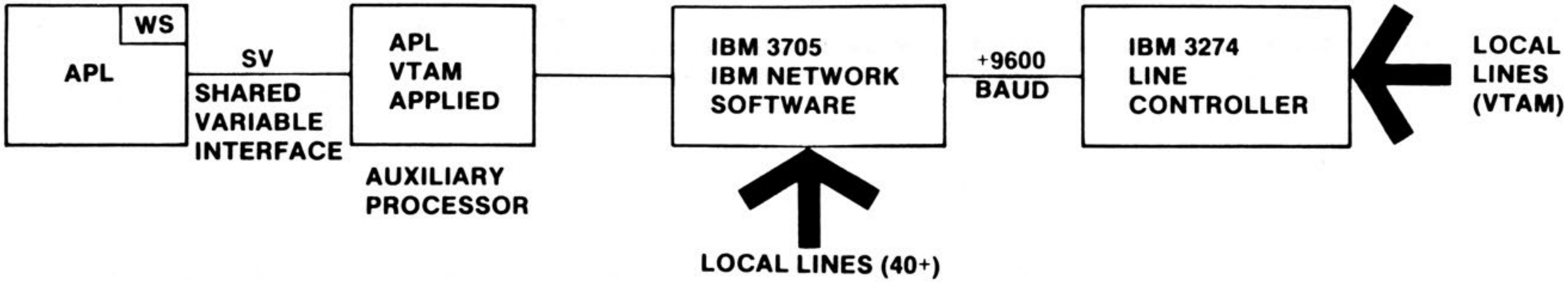


EXHIBIT IV CONVERSION, MIGRATION SYSTEM DEVELOPMENT

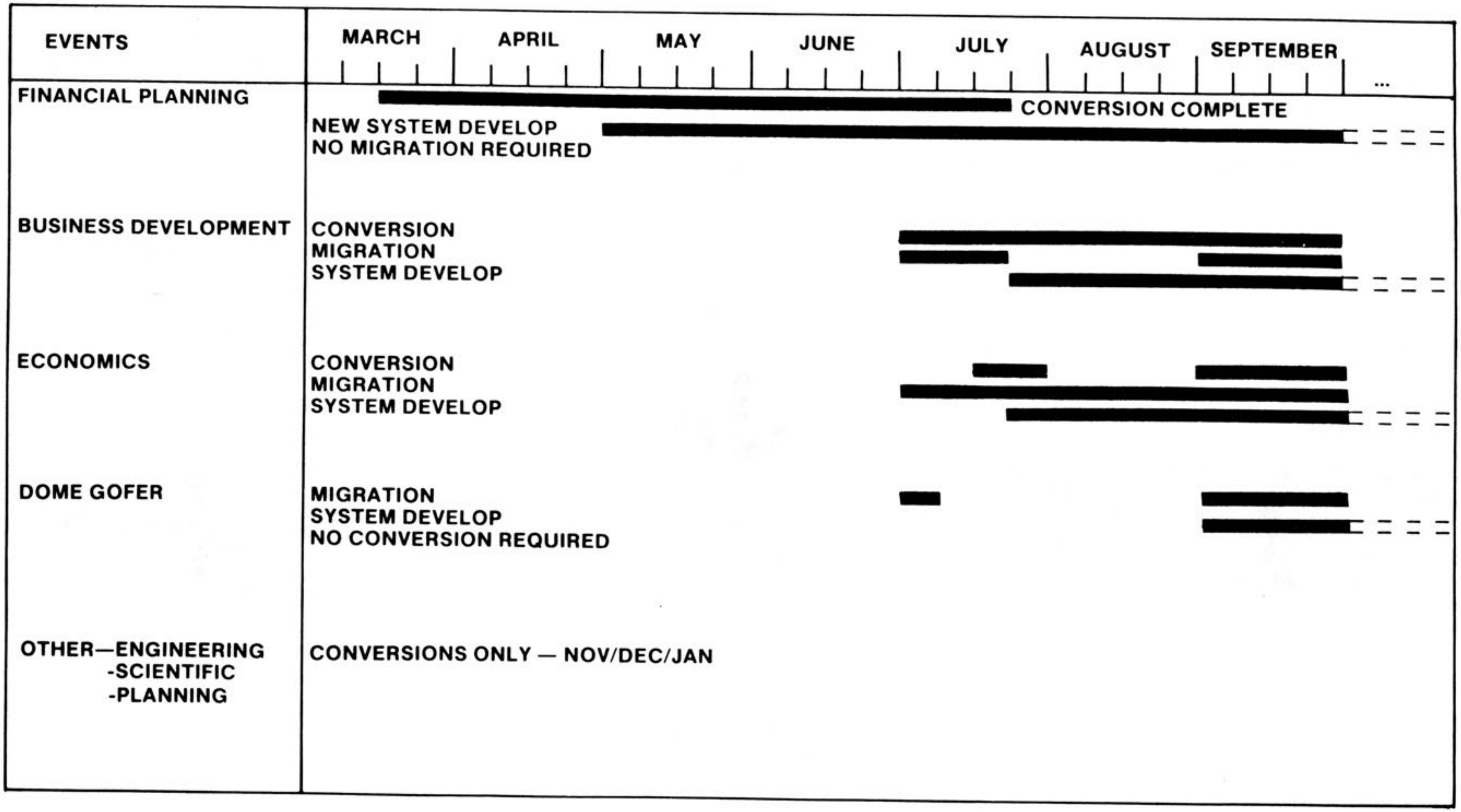


EXHIBIT V Key Reasons That The APL Resource Should be In-House

High Cost - external
Security Exposure

Lack of Integration
Technological Constraints

Control
Growth

1. High Cost - external

IPSA SHARP APL	$60-$120/Hour
Dome SHARP APL	$10-$ 20/Hour

2. Security Exposure
   We do not control the IPSA SHARP APL
   Environment - Operations
   Network Access
   Consultants

3. Lack of Integration of Data and Systems.

4. Technological Constraints
   Telecommunications
   Hardware

5. Control of the resource in which we have made a very significant investment.

6. Growth without excessive cost and with full benefit.

EXHIBIT VI Selection Criteria

(A) IBM vs IPSA
(B) IPSA vs Dome SHARP APL

	(A)		(B)	
Criteria	IBM	IPSA	IPSA	Dome
APL Features (Serviceability)				
APL	Yes	Yes	Yes	Yes
Advanced Function	Weak	Yes	Yes	Yes
File Subsystem	Weak	Yes	Yes	Yes
Host Interface	Yes	No	No	Yes
Research and Development	Weak	Yes	Yes	Yes
User Acceptance	Weak	Yes	Yes	Yes
Conversion Efficiency	Weak	Yes	Yes	Yes
Network Support				
Start/Stop Asynchronous	Yes	Yes	Yes	Yes
VTAM	Yes	No	No	Yes
Bisynchronous Telecommunications	Yes	Yes	Yes	Yes
World-Wide Network	No	Yes	Yes	Yes
Reliability	Weak	Yes	Yes	Yes
Performance (Availability)				
< 5 Users	Yes	Yes	Yes	Yes
> 5 Users	No	Yes	Yes	Yes
Appications Software	Weak	Yes	Yes	Yes
Security	Weak	No	No	Yes
Contract Negotiation	No	No	No	Yes
Test - Verification of Claims	No	Yes	Yes	Yes

EXHIBIT VII

CLASSIFICATION OF USERS BY TYPE AND DEPARTMENT

	2nd Qtr 1980			3rd Qtr 1980			4th Qtr 1980
	A	B	C	A	B	C	Totals
Department							
Bus. Development	5	2	2	5	2	4	11
Corp. Economics	-	3	5	1	4	6	11
Fin. Planning	2	3	—	3	3	—	6
Financing	-	1	1	-	1	1	2
Gas Utilization	1	3	6	1	5	12	20
Oil & Drilling	-	-	2	-	6	10	17
Data Processing	3	-	-	3	-	-	3
Exploration	-	-	-	-	-	-	4
Geol./Geophys.	-	-	-	-	-	-	4
Other	1	-	-	1	-	2	5
	11	12	16	14	21	35	84
TOTALS		39			70		84

Class of User

A = competent programmers that spend a major share of their time on the system or clerical users that are updating/entering or running work for others.

B = programmers who can develop simple programs and who use the statistical, financial, GOFER, etc., packages as well.

C = people who use only the packaged programs such as GOFER exclusively.

PROJECTED GROWTH OF USERS BY CLASS

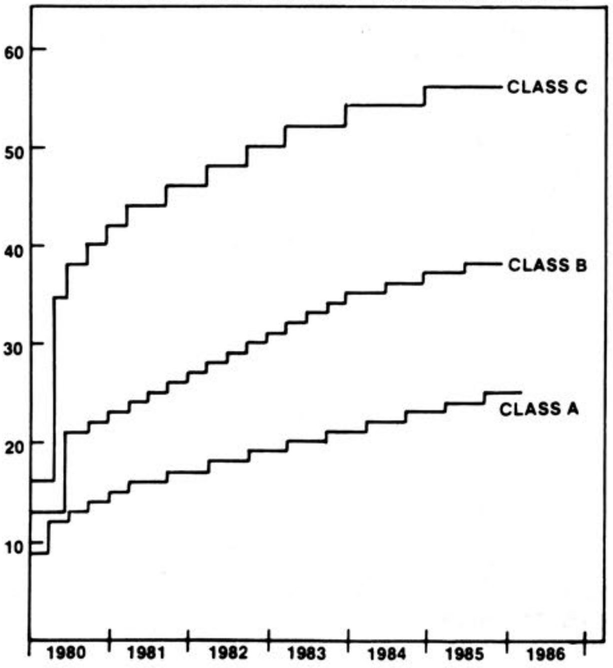


EXHIBIT VIII
PROJECTED USER LOAD AND PEAK DAY CONCURRENT LOAD

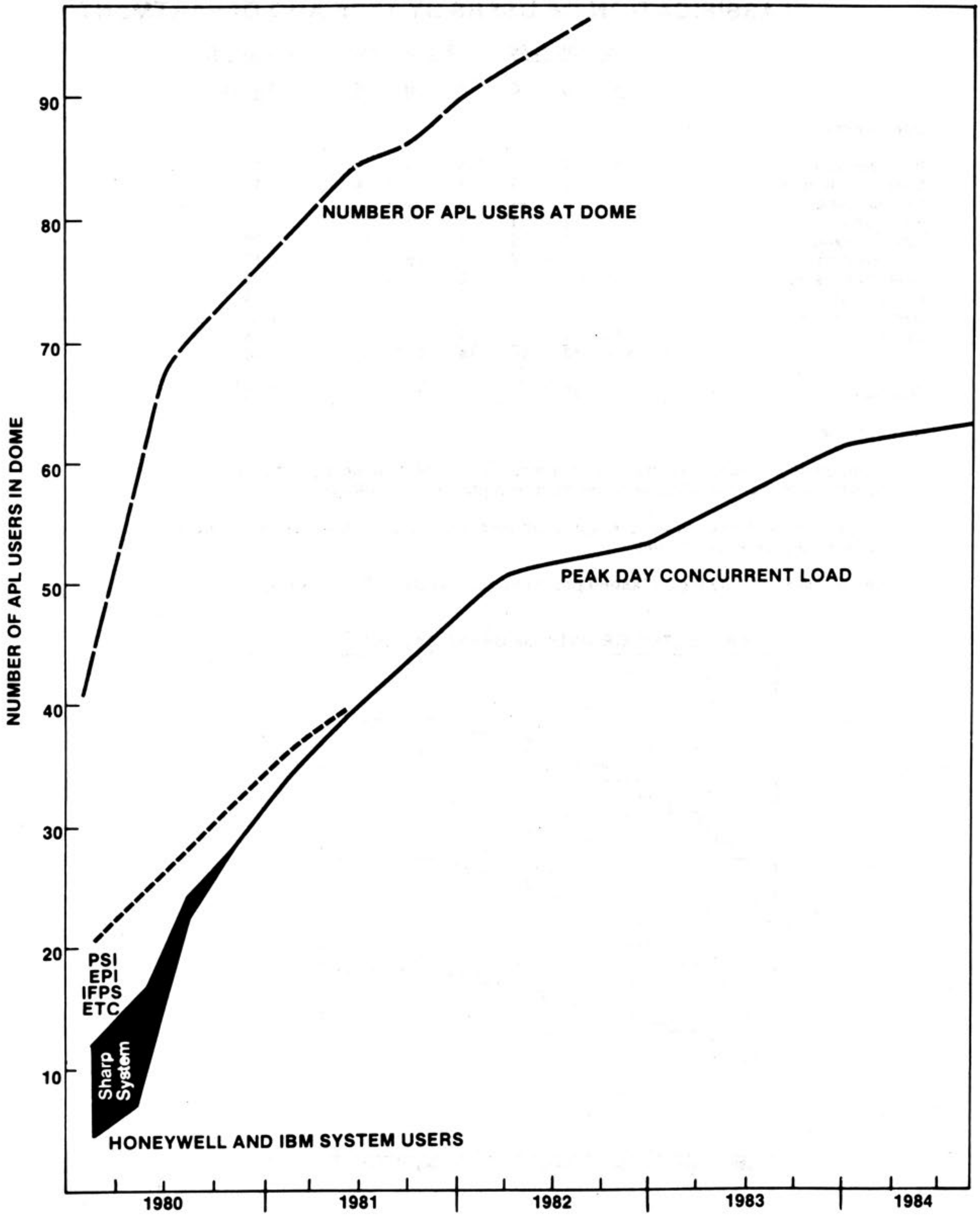
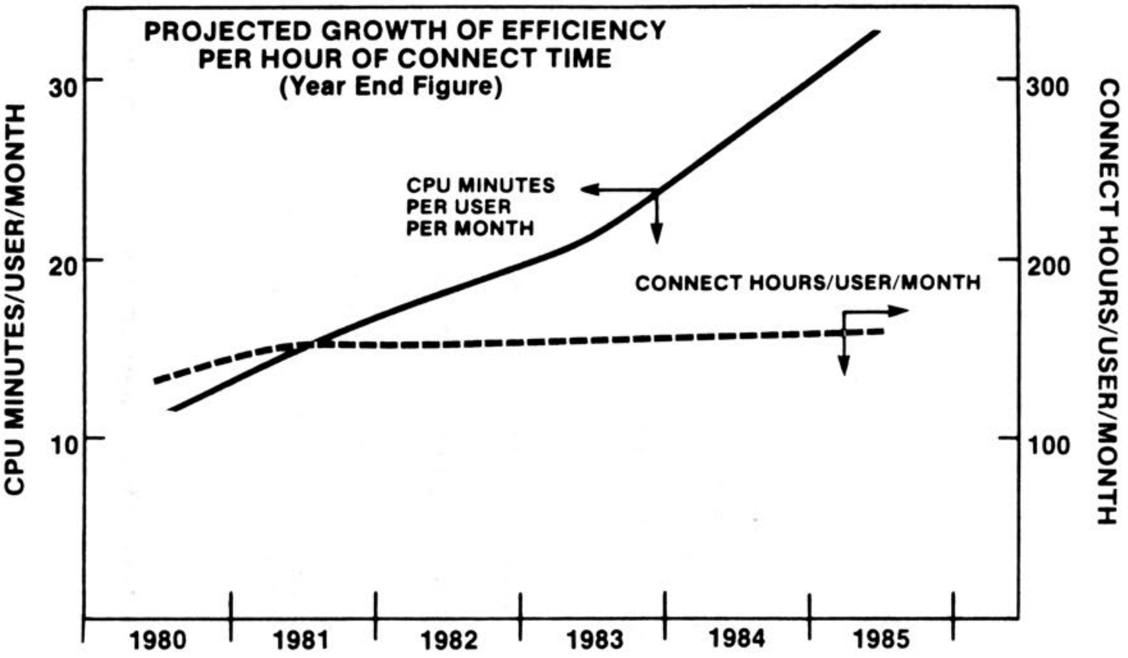


EXHIBIT IX
CONNECT HOURS AND CPU HOURS PER USER/MONTH



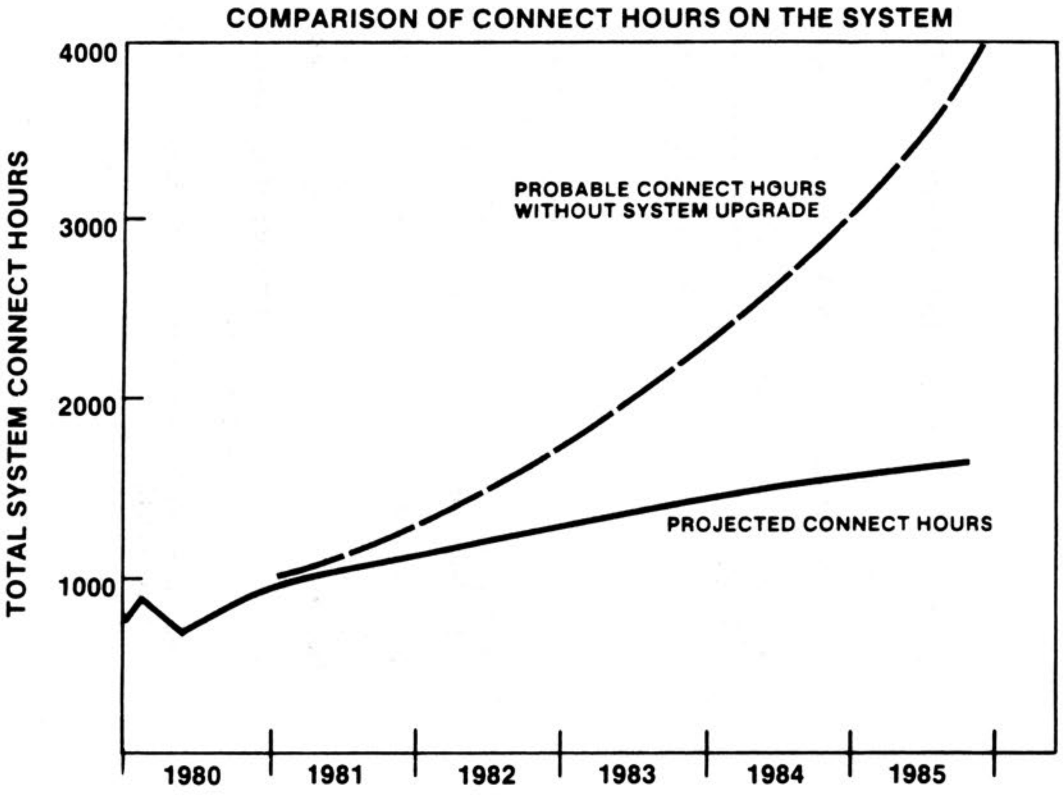


EXHIBIT X Economic Analysis of Dome SHARP APL

We must anticipate a 3-fold to 5-fold growth in our use of APL, this year.

Growth planning is based on the current requirements for APL, the new system development for this year and more effective utilization of the resources made available by the new technology.

PROJECTED GROWTH OF APL AND COST SAVINGS
($000)

	1980	1981	1982	1983	1984	1985
I.P. SHARP (after discount)	802	1,112	1,301	1,508	1,676	1,825
Dome SHARP APL	838	836	1,001	1,224	1,442	1,658
Net (bt)	36	276	200	284	234	167
Net (at)	(19)	146	106	151	124	89

PV (bt) $728,000 at 12% discount rate
PV (at)$465,000

The initial capital investment is $100,000 for installation services provided by I.P. Sharp Associates. The internal rate of return is 510%.

EXHIBIT XI
PROJECTED GROWTH OF APL SYSTEM AND COST SAVINGS REALIZED

PARAMETERS	1980	1981	1982	1983	1984	1985	
No. of Users (4th Qtr)							
Class A	14	17	19	21	23	25	
Class B	23	27	31	35	37	39	
Class C	40	46	50	52	54	56	
Total	77	90	100	108	114	120	
Max. No. on Line (4th Qtr)	42	49	55	62	65	68	
Connect Time (4th Qtr)	975	1,098	1,290	1,414	1,508	1,602	
Cost/Hr @ Sharp ($)	112	127	143	161	181	203	
Sharp							
Total Cost @ Sharp ($K)	1,069	1,598	2,002	2,514	3,047	3,649	
less Volume Discounts	267	476	701	1006	1,371	1,824	
New Cost @ Sharp	802	1,112	1,301	1,508	1,676	1,825	PVbt 12% 728
							PVat 12% 386
Dome							
Computer Charges ($K)	173	261	336	443	583	770	
Connect Charges	190	200	205	211	214	218	
Support Staff	120	155	200	260	285	240	
Outside Services	80	100	110	120	130	140	
Hardware Costs	45	70	90	120	150	200	
Other	70	50	60	70	80	90	
	728	836	1,001	1,224	1,442	1,658	
Potential Savings ($K)	124 (full year)	276	200	284	234	167	
(over Sharp Service)	72 (partial year)						
less Installation Costs	(108)						
Net Cash Flow (bt)	(36)	276	200	284	234	167	
Net Cash Flow (at)	(19)	146	106	151	124	89	

MAKING APL PROGRAMMERS MORE PRODUCTIVE

Robert C. Metzger
I.P. Sharp Associates, Inc.
Rochester, New York

Introduction

People who use APL seem convinced that it improves their productivity. People who sell APL software or services make bold claims of productivity improvements. Some say 3 to 1, some 5 to 1, others even 10 to 1. There are, of course, many reasons why this should be so. But, if these claims are true, why isn't everyone using APL?

There are several reasons for the fact that APL is still only used in a small minority of computer applications. One is historical inertia. People will do tomorrow what they did today because that's what they did yesterday.

A second, but rarely admitted, reason is that APL programmers are not as productive as they could be. The first part of this paper explores ways of avoiding several pitfalls which can nullify APL's productivity advantage.

These pitfalls are:

1. Inadequate training,
2. Isolation from other APL programmers,
3. Inefficient implementations,
4. Incorrect methods,
5. Lack of programming standards,
6. Inadequate tools.

The second part of this paper explains tools and techniques which can greatly enhance the effectiveness of APL programmers.

These are:

1. Idioms,
2. Pseudo-primitives,
3. Building blocks,
4. Templates,
5. Generators,
6. Implements,
7. Documentors.

Training

A large number, if not a majority, of APL users are amateur, rather than professional. programmers. That is to say, many APL users are writing APL programs as an incidental part of their job responsibilities, which lie in a field other than computers. That this is possible with APL is one of the chief reasons for its popularity.

This advantage becomes a problem when the amateur programmer moves into the field of the professional. As Weinberg has pointed out, "Almost invariably, the sole intended user of an amateur's program is the amateur himself, whereas the professional is writing programs which other people will use." [1] The skills needed for producing systems for others are different than those required for doing your own work.

> We might call this the Fallacy of the Five Day Class:
>
> "A person who has no computer background can be taught in a week to write APL programs for live applications. THEREFORE, that same person can analyze, design, and build applications for others with no further training."

It's just not true.

Many organizations use APL because it allows them to create and change programs rapidly. Such organizations often are in a business which changes quickly. They may also see frequent changes in their own organization. Many of these suffer from what I call the ASAP Attitude:

> "We don't have time to give our people training."

Training can help a person write programs which are easier to use and less expensive to run. If you don't have time to do it right the first time, you'll just end up doing it again.

Besides university courses, timesharing and consulting companies often offer courses of great value to the APL programmer. Books are perhaps the most overlooked source of training. Many good books of recent vintage are produced by some of the same people who teach courses. I would be happy to provide anyone with a list of 10 - 20 books I wish every APL programmer would read. The fact that APL is easy to use does **not** negate the need for training.

Isolation

APL users frequently do their work alone. One reason is that they are often doing work for themselves. Even when they are producing a system for someone else, multi-person projects are rare. APL makes people productive enough that major tasks can be done by a single person in a reasonable time.

This causes a problem of isolation. It is not uncommon for an APL user to be the only person in a department who knows programming. There are several bad effects. There is a tremendous amount of wheel re-invention. Isolated users are unaware of software developed by others which is already available. People who haven't had formal training in computer science are often not aware of common concepts, algorithms, procedures, etc. Isolated users don't have anyone to talk over problems or design ideas. "Two heads are better than one" is certainly true of systems design work.

How can isolation be overcome? There are a number of possibilities.

1. Provide a computer book budget for the APL user,

2. Obtain a subscription to the APL Quote Quad,

3. Send the user to advanced training courses,

4. Send the user to APL conferences. (Certainly one of the reasons APL conferences are so popular is that many people feel, "At last I've found some people who know what I'm talking about.")

Implementations

Inefficient implementations are a terrible strain on productivity. They force people to write programs in unreasonable ways to get reasonable performance. Such problems occur when it is possible to write a defined function which is faster than a primitive which produces the same result.

The way one implementation handles the scan operator is an example of such a problem. There are two (2) possible algorithms for computing scans of associative functions. This implementation does not recognize the fact that ≠\ (NOT EQUAL SCAN) is associative with boolean data. It, therefore, uses a very poor algorithm. The algorithm for the primitive uses !*N* comparisons, while a defined function can be written which does only N comparisons. For arrays of any significant size, the APL function, which requires a loop, is much faster.

If you are using such an implementation, you have two choices.

1. Pressure the vendor into fixing things.
2. Find another vendor.

Inefficient implementations steal your money when you are writing programs for them, and when you are running them.

Not only can implementations be inefficient, but they can also be poorly human engineered. Those implementations which provide only the ⍕ (thorn) function for report formatting are a good example. This function does not provide for any of the output formatting features which are absolutely necessary in a business environment:

1. the ability to substitute '-' for '¯' with negative numbers;
2. the ability to insert commas between every 3 digits;
3. the ability to prefix and suffix negative numbers (with parentheses or debit/credit notations);
4. the ability to blank out zero values.

As a result, the programmer must build code into every report to do these things. Such wasted effort is completely unnecessary. What's the point of using APL if you have to simulate features which belong in the system software?

Incorrect Methods

Inadequate training, isolation from other APL programmers, and experience with other programming languages are the chief reasons for the use of incorrect methods. The first two reasons have been discussed. The third may seem quite surprising.

Traditional languages like FORTRAN, COBOL, PL/I, ALGOL, etc. are scalar oriented. Since only a single piece of information can be processed at a time, explicit looping is both necessary and effective. APL, however, is array oriented. Because current APL implementations are interpreted, rather than compiled, explicit looping can be expensive.

There are actually only a few situations when explicit looping is necessary in APL. The first is when you are using a poor implementation which executes defined functions faster than primitives (see above). The remainder occur when all the data to be processed is not in the active workspace at the same time. This can happen in several different ways. All of the data may not fit in the active workspace. So it will be stored in a file and processed in parts. Or all the data may not be available because it is being given to the APL program in parts in real-time. This happens when a person at a terminal or another task is providing input. It is unusual to write loops in APL if one of these conditions does not hold.

If a programmer uses valid methods from another language, he is probably using loops. If he is using loops, he is writing code which could be more concisely expressed in APL. The end result is programs which take longer to write and longer to run. APL applications require APL methods.

Programming Standards

APL is a rich notation. There are, therefore, many good APL ways of solving most problems. This richness has led many APL enthusiasts to excesses of obscure coding, all in the name of "freedom" and "artistic license". If your APL usage is entirely recreational, this is fine. If you use APL to accomplish work to benefit students, employers, stockholders, or whatever, this is inexcusable. This paper addresses the issue of productivity because productivity is important. Standards increase productivity.

Any programming standard constructed in a rational manner by a person knowledgeable in APL will have at least three (3) benefits.

1. Programmers will write programs faster. This is because they will write in whole phrases, rather than symbol by symbol. This latter situation happens when a person must make a conscious choice of expression at every point.

2. Programmers will read (and understand) programs faster. Familiar styles and conventions are like landmarks along a road. They point the reader in the right direction. They help him ignore details which are not of concern.

3. Programs will have fewer bugs. The whole point of standards is to consistently use tried-and-true methods. Reliable methods produce reliable programs.

Listed below are some of the important issues which any APL programming standard should address.

1. Statement Syntax
   a. Embedded assignments;
   b. Statement gluing (,0ρ) and statement separators (◇);
   c. Obsolete constructs (IBeams, mixed output).

2. Program Form
   a. Maximum or average number of statements per function;
   b. Maximum or average number of characters per line;

3. Program Logic
   a. Naked branch (→);
   b. Branches using absolute line numbers;
   c. Branches using ⎕*LC*;
   d. Preferred forms of logic constructs;
   e. Conditional execute.

4. Naming Conventions
   a. Functions vs. variables;
   b. Global variables, semi-global variables, and local variables;
   c. Line labels;
   d. Use of delta, numeric digits, and underscored alphabetics;
   e. Use of acronyms, abbreviations, and single letter names.

5. Input/Output
   a. Arguments and results;
   b. Terminal input;
   c. Printed output;
   d. File input/output.

6. Comments

The importance of APL programming standards cannot be overestimated. Standards are to programming what practice is to a musician. Only through much discipline can true artistic freedom be expressed. This is a hard lesson to learn, but freedom without discipline is anarchy.

Software Tools

An excellent book entitled **Software Tools**, by Kernighan and Plauger [2] provides some background to the discussion of APL programming tools. The concepts presented in this book are quite valuable. Unfortunately, the programs explained are not very useful to APL'ers since they are written in a dialect of FORTRAN called RATFOR.

Through many examples, Kernighan and Plauger demonstrate the three types of software tools they distinguish. The first of these are what they call filters. A filter takes 1 input and produces 1 output, which is some useful transformation of the data. The second type is what they call a primitive. A primitive hides details of the host environment. It makes groups of programs more portable, since primitives whose external behavior is the same can be written for different environments. The third category are programs which develop programs, which is what the book is all about.

Kernighan and Plauger present several helpful suggestions on how to build programming tools.

1. Build more complicated tools from simpler ones (pyramiding).

2. Push details as far down as possible (generality).

3. A change in one function's algorithm shouldn't affect another (low degree of coupling).

4. Keep functions separate until you know how to combine them (independence).

5. Programs must be able to work together (the whole is greater than the sum of the parts).

6. Break larger programs into smaller pieces which communicate only through well-defined interfaces (functionality).

The authors use two concepts which are not directly relevant to APL. The primitive data structure they use is a file, while it is the rectangular array in APL. The program interface they use is the pipeline notation, while we use arguments and results in APL. The book as a whole, however, gives a very helpful view of software tools from the perspective of another language.

APL Software Tools

APL programmers are much more productive when they have a comprehensive set of programming set of tools available to them. Just as people use APL as a tool in other disciplines, so the APL programmer must have APL techniques and tools to help him build APL programs.

There are a number of ways to categorize APL programming tools. McAuley and Nelson [3] divide APL programming tools into three classes:

> Utilities, which accomplish routine tasks like entering, formatting, and manipulating data.
>
> Documentors, which display and analyze pieces of a system.
>
> Function fixers, which modify the elements of the system.

Phil Abrams [4] distinguishes between utilities, which can become part of an application, and tools, which are used to prepare application programs.

Abrams' categories are useful, but the terms have such varied usage that I prefer "internal tools" and "external tools". The remainder of this paper explains the uses of different kinds of internal and external tools.

Idioms

The concept of an APL idiom is due largely to Perlis [5]. He identifies four properties of idioms:

1. frequency of occurence,
2. unity of purpose,

3. ease of recognition,
4. composability of use.

An idiom is a single expression which performs a single operation. Some people put idioms into one-line functions. They are often more useful embedded as a part of larger expression.

Mastery of a standard set of APL idioms is essential to productive APL programming. The reason is "once they are learned, they are thereafter recognized as a unit". [6] A person who is learning to read a natural language starts by processing one letter at a time. As he progresses, he moves up to syllables, words, and phrases. The same is true of writing in natural languages. If you are fluent in a language, you write in phrases or idioms, not word by word. If you have ever watched a very productive APL programmer, you will notice that he probably writes whole lines at once. Each line contains one or more idioms.

The reason that idioms are so powerful has been explored by Dr. Ben Schneiderman. He would call APL idioms "low level semantic structures".[7] His "syntatic/semantic model of programmer behavior" suggests that experienced programmers learn to recognize semantic structures, and thus read and understand programs more quickly. He says that programmers "might be taught to look for familiar statement patterns or templates....Program composition should be taught by the development of templates which are organized into modules, analagous to paragraph formation. This approach generalizes the top-down modular design techniques proposed by structured programming advocates" [8].

Idioms can be grouped into several categories.

1. Structural,
2. Sorting and Searching,
3. Formatting,
4. Predicates,
5. Boolean Scans,
6. Boolean Shifts,
7. Numerical,
8. Object Generators.

Some examples are given below.

1. Structural. Make a matrix from a scalar, vector, or matrix.

```
(1⌈¯2↑⍴ARRAY)⍴ARRAY
```

2. Sorting and Searching. Sort a vector according to a sequence.

```
VECTOR[⍋SEQUENCE⍳VECTOR]
```

3. Formatting. Double space text.

```
((2×⍴TEXT)⍴1 0)\TEXT
```

4. Predicates. Is this an empty array?

   ```
   0∊⍴ARRAY
   ```

5. Boolean Shifts. Give a one where the input is one or the previous element in the input is one.

   ```
   B∧¯1↓0,B
   ```

6. Boolean Scans. Give a one between successive ones.

   ```
   (~B)∧≠\B
   ```

7. Numerical. Multiply vector times columns of matrix.

   ```
   MATRIX÷⍉(⌽⍴VECTOR)⍴VECTOR
   ```

8. Object Generators. Create identity matrix.

   ```
   (⍳N)∘.=⍳N
   ```

Idiom lists can be found in Perlis' paper cited above, or in WS 881 UTILITY on the SHARP APL Timesharing Service.

Pseudo Primitives

These tools are on the next level up in the taxonomy of tools. Like idioms, pseudo primitives are filters, in the scheme of Kernighan and Plauger. They are defined APL functions. Their input comes only from their arguments, and their only output is a result. They are internal tools. They become part of the final software product.

The first difference between idioms and pseudo primitives is length. Pseudo primitives normally take more than one statement to implement. It is thus easier to call them as separate functions.

The second difference is their relationship to existing primitives. Pseudo primitives are either variants of existing APL primitives, or functions which are complementary to APL primitives. Some of them have been implemented as primitives, and they are a prime source of suggestions for new primitives.

The following examples illustrate some of the more commonly used pseudo primitives.

1. String search. A variant of dyadic ⍳ in which searches are done for a vector as a group, rather than element by element.

   ```
         'THE CAT IN THE HAT CAME BACK' ∆LOCATE 'CA'
   0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
   ```

2. String replace. A variant of indexed assignment in which the objects to be replaced are specified by their value, rather than their location.

   ```
         'THE CAT IN THE HAT CAME BACK' ∆SREPLACE '/AT/OW'
   THE COW IN THE HOW CAME BACK
   ```

3. Matrix from vector. A variant of reshape in which the structure is determined by the values contained, rather than being explicitly defined.

```
      ';' ΔMFV 'ZEBRA;GIRAFFE;LION'
ZEBRA
GIRAFFE
LION
```

4. Join arrays. A variant of catenate/laminate in which the array which is smaller along the axis being joined is extended.

```
      ZOO←';' ΔMFV 'ZEBRA;GIRAFFE;LION'
      ZOO ΔJOIN 'ELEPHANT'
ZEBRA
GIRAFFE
LION
ELEPHANT
```

Pseudo primitives must be carefully designed for generality and carefully coded for efficiency. They are essential for building more advanced tools.

Building Blocks

Building blocks differ from pseudo-primitives in one major respect. Their input sources and/or output targets are not rectangular arrays. They normally take arguments and return results. These arguments and results, however, are more often control information than data to be transformed.

Building blocks are what Kernighan and Plauger call "primitives". Frequently, they hide details and implementation or hardware dependencies. They are internal software, and so are often incorporated into application packages.

Building blocks fall into several categories:

1) Interactive input;
2) Terminal control;
3) Output formatting;
4) File handling, sorting, printing;
5) Data base management;
6) Data conversion from non-APL sources.

One of the chief advantages of using an APL timesharing service is that the good ones provide such software as a part of their services. They have learned through the experience of building applications how important such tools are.

Templates

Templates are skeleton programs. They contain general code which must be augmented or altered to fit specific situations. Some templates are fleshed out with a general purpose program. The convention is adopted that a comment which consists only of a del is a "blank" to be filled in. Other templates require special programs to fill them in.

Templates are internal tools. They become part of the final product. The programs which manipulate templates are external tools, since they do not become part of the final product. Templates are related to the "macro" concept in other languages.

The function listed below is a template. It loops through a file, reading some or all of the components. What it does with them depends upon what is put between lines [8] and [10].

```
     ∇ FILELOOP TI̲E;CT̲R;LI̲MI̲T;CO̲MP̲
[1]    ⍝∇
[2]    CT̲R←0⌈¯1+1↑1↓TI̲E
[3]    LI̲MI̲T←1↑2↓TI̲E
[4]    TI̲E←1↑TI̲E
[5]    LI̲MI̲T←(⍲\2⍴ LI̲MI̲T=0)/(¯1+1↑1↓⎕SIZE TIE), LI̲MI̲T
[6]    LO̲OP̲:CT̲R←CT̲R+1
[7]    →(LI̲MI̲T <CT̲R)/EN̲D
[8]    CO̲MP̲←⎕READ TI̲E,CT̲R
[9]    ⍝∇
[10]   →LO̲OP̲
[11]   EN̲D:
[12]   ⍝∇
     ∇
```

If line [8] becomes

```
[8] ⎕←CO̲MP̲
```

the function displays the contents of the file.

If lines [1], [8], [12] are defined as follows

```
[1] RO̲WS̲←0
[8] RO̲WS̲←RO̲WS̲+1↑⍴CO̲MP̲
[12] ⎕←'ROWS= ',⍕RO̲WS̲
```

then it counts the number of rows of the matrices in the file. The possibilities are endless.

The chief advantage of a template is that you can make it a thousand different programs. It's all debugged, and can be custom tailored to fit your needs. It saves you time.

Generators

Generators are very sophisticated tools. They are external tools. They create, without using a template, programs which will become part of the application product. They are related to macro processors. They are more powerful because the full APL language is available. This is not true of macro processors for languages like PL/I and some assemblers.

The following example illustrates a generator. A typical APL application is a financial model in which the conceptual data structure is a 5 dimensional array. The dimensions are:

[1] Type (plan or actual)
[2] Year (previous, current, next)
[3] Locations (many)
[4] Products (many)
[5] Month (1-12)

Unfortunately, the length of the first two dimensions would require a matrix which would occupy a megabyte or more. Implementation or installation constraints make this impossible, so the data will be stored in a file as matrices created from the last 2 dimensions.

Much of the output for this system consists of reports which either total (+/) or summarize (partitioned +/) along one of the dimensions of the conceptual array.

If we have a generator which produces reduction functions for such a conceptual array, we just need to type

```
      '+' BUILDΔREDUCE 5
```

and the generator will create 5 programs, each of which will sum reduce the array along a different axis.

If you frequently use similar file structures for different applications, it may be worthwhile to you to write a generator which will create the necessary file cover functions. Other applications can also benefit from this technique.

Utensils

Programming utensils help the programmer in the various steps of the development process. They handle or manipulate functions or data so as to reduce programming effort. They are external tools, and do not become part of the target system. They may not even directly produce code.

The simple utensils are the most commonly used.

1) *FNBUILD* - build a function from a template
2) *FNCOPY* - make a copy of the code, with a new name
3) *FNMERGE* - put 2 or more functions into 1 function
4) *FNSPLIT* - break up 1 function into 2 or more functions
5) *FNCOMPARE* - compare 2 functions

More complicated tools include sophisticated program changing and improvement [9]:

1) edit functions,
2) search, display, and change texts in a group of functions,
3) localize names,
4) alphabetize locals in header,
5) breakup embedded assignments,
6) unblock grouped statements,
7) restructure branch statements,
8) convert obsolete language constructs,
9) rename variables,

10) rename labels,
11) rename comments.

Such tools are essential to the rapid production and maintenance of application programs in a dynamic environment.

Documentors

Documenting tools aid the programmer during initial development and subsequent maintenance. They are external tools, and don't usually produce code.

Some standard documentors are listed below:

1. Function definitions,
2. Function cross-references,
3. Variable definitions,
4. Workspace cross-references.

More sophisticated documentors are often used in debugging [10].

1. Display expanded program listing,
2. Display program aligned in columns,
3. Program flowcharts,
4. Highlight syntactic elements of program using special display features of terminals.

Programs can also be developed which aid the production of user documentation, in addition to that for programmers.

Conclusion

The full potential of APL has only begun to be reached. The methods and philosophies of old situations and different environments must be abandoned. APL programmers must be demonstrably more productive if Iverson's "better mousetrap" is going to replace the worn-out languages of the DP world.

APL programmers must seek ever better software tools to bring about productivity increases. The tools described in this paper are a comprehensive set of aids to programmer productivity. They should be available to all APL users, and all APL users should have education in their use. In building application systems, APL is the finest raw material. APL users also need the finest tools to manipulate that raw material. A craftsman is only as good as his tools.

References

[1] Gerald Weinberg, **The Psychology of Computer Programming**, (New York: Van Nostrand Reinhold, 1971), p. 122.

[5] A. Perlis and S. Rugaber, "Programming with Idioms in APL", **Proceedings of the APL79 Conference**, (New York: ACM, 1979), pp. 232-235.

[6] Perlis and Rugaber, ibid, p. 232.

[2] B. Kernighan and P.J. Plauger, **Software Tools**, (Reading, Mass.: Addison-Wesley, 1976).

[4] P. Abrams, "Large Applications in APL", speech given at the APL80 Conference, June 25, 1980.

[3] S. McAuley and J. Nelson, "The Professional Programmer's Tool Kit", **APL in Practice**, (New York: Wiley-Interscience, 1980), pp. 339-344.

[8] Ben Scheiderman, **Software Psychology**, (Cambridge, Mass.: Withrop Publishers, 1980), p. 32.

[7] Ben Scheiderman, "Perceptual and cognitive issues in the syntactic/semantic model of programmer behavior", **Proceedings of Symposium on Human Factors and Computer Science**, Human Factors Society, Santa Monica, California, 1978.

[9] Robert Metzger, "A Toolbox for APL Programmers", **Proceedings of APL79 Conference**, (New York: ACM, 1979), pp. 236-243.

[10] Metzger, ibid.

DECISION SUPPORT SYSTEMS:
THE POWER AND PROBLEMS OF APL

Maurice Elliott
Dome Petroleum Limited
Calgary, Alberta

Abstract

This paper discusses the evolution of computers and systems over the past 35 years or so, to the point where many organizations now routinely expect computer systems to support the decision-making roles of their senior executives. It describes the characteristics required of these systems, and examines the strengths and weaknesses of APL as a language for implementing them. A discussion of software tools and management steps to minimize the impact of APL's weaknesses concludes this paper.

Introduction

Over the 35 years or so that computers have existed, we have seen a breathtaking evolution in their capabilities and in people's expectations of them. Hardware speeds have increased by leaps and bounds while costs have plummetted. On the other hand, the cost of the people who make the machines effective soared as inflation and the need for specialist skills took their toll. Thus it has been increasingly justified to make the machine work harder in order to save people-time. We have seen high-level languages, sophisticated operating systems, database software, and pre-programmed application packages, all helping people to be more productive at the expense of machine time.

Applications have seen their share of the evolution, too. The early ones were specific numeric computations, often programmed, tested, and run by one man on a dedicated machine. Then businesses began to use computers to mechanize repetitive clerical work. Again the systems were relatively simple, were developed by small teams for one department, and enjoyed exclusive use of the machine. Because of their simplicity, they were usually produced quickly, despite the primitive hardware and software (by today's standards) that was available.

As management demanded more and better information, and tighter control of the business, applications became more complex. They crossed departmental boundaries, and since a system failure would have more serious consequences for the business, they had to be more reliable.

In recent years, management expected systems to play a larger role in their decision-making and planning activities. The path from the clerk's desk to the executive suite has been far from smooth, however, and many companies have found it very difficult to adapt their systems to this new role. The remainder of this paper discusses these difficulties, and the reasons why APL can shine in this environment. Some counterproductive aspects of APL are also examined, and ways to minimize their impact are suggested.

The Decision-Making Environment

Why is it that systems, which seemed to be coping so well with clerical work and routine reporting, have found it so hard to win the confidence of decision-makers? The answer is that the character of their work is vastly different. Clerical work consists largely of repetitive tasks and decisions which can be anticipated and the appropriate responses pre-defined, using available quantitative data. It is concerned with facts about events and decisions which have taken place, and whose impact can be measured. The number of facts may be large, but their variety and the ways in which they are recorded and analyzed is sharply constrained by the goals which the organization strives to achieve, and the frames of reference (internal or external) within which it chooses to operate.

Decision-making, on the other hand, is much more concerned with the goals and frames of reference themselves. It seeks to modify them to improve the operating results of the organization. Of course, the decision-making process is itself bound by constraints (legal, moral, political, etc.), but they are generally much wider than those governing clerical work. The most important parts of any decision are in fact the identification of the real problem to be solved, and the question of implementation strategy. And in both of these, interpersonal and inter-group relations are paramount considerations.

Any computer system which ventures into this arena is faced with two fundamental hurdles: the nature of the task is creative in that it explores new ground-rules; and the essential information is largely qualitative. These hurdles obviously cannot be cleared without great effort. Indeed, we may have to wait for the fruition of much of today's AI research before we can progress very far. Meanwhile, we must look for solutions which make the best use of the strengths of both the human decision-maker and the machine.

Compared to machines, humans are superbly creative and flexible. They can crystallize out the essential features of an ill-defined problem-situation, perceive what information is pertinent and seek it out, dream up innovative solutions, spot flaws in arguments, weigh alternatives, discriminate between promising and unpromising leads, recognize dead-ends, switch lines of attack when stalled, and muse about a problem in general. The machine is totally devoid of the imagination and sensitivity which are vital to the activities. It must be given a precise unambiguous set of rules, and quantitative data to work with. But the machine, once it has been instructed and fed with data, leaves the human standing when it comes to following a procedure quickly, flawlessly, and without fatigue time after time.

What we must do is free the human for the creative excursions while eliminating the mechanical drudgery of evaluating or estimating the impact of the new ground rules and parameters that emerge. Our support will be much more valuable if it encourages the decision-maker to explore alternative, perhaps dissenting, views of the problem. For it is this exploration that, if used properly, can show how to turn a mediocre proposal into a good one, and a good one into a better one. The "right" decision is a pipe-dream in most situations — the most we can hope for is to make more better decisions than the competition.

The speed of the computer is vital to the achievement of this aim. But it is not sufficient. We must also have the ability to reliably and quickly turn the computer around in order to examine newly-formulated proposals. Lack of flexibility in our systems is the largest single obstacle to the use of computers by decision-makers and their staff today.

The Power of APL

How does APL stack up as an implementation language in this environment? To answer this question, let's examine some characteristics of the language.

First, APL is a truly conversational language. It was originally designed to be responsive, and until recently, there was no other way to run an APL program than interactively through a terminal. Given a suitably-designed system, the user can get very quick answers to his questions. This is crucial if a decision-maker's probing is to be supported and encouraged, and not hampered by the system.

Second, well-thought-out APL programs are very compact due to the concise notation and powerful instruction repertoire. This means the coding stage of system development can proceed much faster than with more verbose languages. The design stage, too, is shortened because many common non-trivial functions (such as in-core sorting and table searching) are built into the language and do not need to be re-invented.

APL encourages modularity. The overhead to build a large number of small modules, compared to a small number of large ones, is minimal. And if the cohesion within modules is high and the coupling between them is low, testing is simplified. APL makes no distinction between main programs and subroutines, so any module can be tested independently, requiring only that a suitable environment and its subordinate functions (or stubs) exist.

APL's reaction to errors is also convenient. Instead of cancelling your job and dumping memory, APL holds everything, prints an error report, and hands control over to you. You could change variables, define new ones, create missing functions, and in many cases even change the definition of the suspended function, in order to fix the problem. Then you could resume execution, at the point of suspension, at any other point in the suspended function, or indeed at any point in one of its callers. Because of this, one test run often eliminates many bugs in an APL system, where the first would have led to cancellation of a batch test. And all of this is done in terms of APL itself. Core dumps, hexadecimal, or whatever are a thing of the past.

APL has no JCL. Everything you do in the APL environment takes the form of an APL expression or system command. Only those of you who have wrestled with operating systems to coax them to compile and run your tests can appreciate the significance of this statement.

Even the compile process does not exist in APL — it executes your source code directly. Thus you can begin testing a function seconds after you have entered the last line of its definition. Small but still not insignificant functions can be conceived, written, and tested literally in minutes by an experienced programmer. And once created, they can be used anywhere the task they perform is required.

The Problems of APL

I have already mentioned the powerful instruction-set of APL. This, unfortunately, is a two-edged sword. Undisciplined programmers can misuse this power to create obscure, totally unmaintainable code. Even a function with no branches in it can be so jam-packed with references to built-in or defined functions as to defy human comprehension in any reasonable time-span. This can occur with any language, of course. But the availability of non-trivial built-in functions, and the ease with which functions can be made to interact, paves the way to this trap in APL. The only real answer to this problem is to enforce programming standards which demand clear, readable code.

Despite its wide repertoire of built-in functions, APL has no direct equivalent of the control structures now considered necessary for building maintainable programs. The IF-THEN-ELSE, DOWHILE and other constructs must instead be simulated using plain GOTO's which Dijkstra's letter to the ACM in 1968 roundly condemned, and rightly so in my opinion. Proponents of APL claim that this is not a problem, since APL's array handling capabilities all but eliminate the need for branching. This is true when the programmer has enough experience and self-discipline to avoid the unnecessary GOTO's, when the cost (in CPU time) of the application is not a concern, and when filing handling is not required. But all the significant systems I've worked on were written by real people, could not be allowed to use unlimited CPU time, and processed the files of data. And few of them were a pleasure to maintain. This criticism even extends to the physical layout of the code. If you take the trouble to indent your instructions as you enter them, for example to reflect the scope of loops, then APL neatly realigns them all to the left margin for you! As long as the implementers of APL continue to bury their heads in the sand on this score, the only defence we have is to establish standards and hire programmers who have the necessary talent and professional attitude to structure their code well despite the faults in the language.

Another problem with APL, again the "back-side" of a two-edged sword, is its dynamic nature. One can write whole systems without making any assumptions about the dimensions of the data they will process. Nor, in theory, need one place any limit on these dimensions. Unfortunately, in practice the programmer must spend far too much of his time worrying about managing the contents of the workspace. It's virtually impossible to anticipate where the dreaded *WS FULL* will strike next, especially if the user has control over the dimensions, as well as the content, of his data. The *WS FULL* problem is exacerbated by the habit of some interpreters (Sharp included) of making a copy of any variable passed as the argument of a function, even if the called function doesn't change it. Programmers hesitate to pass large variables as arguments, and this leads to greater obscurity in the code. Worse still, SHARP APL has no built-in facilities for monitoring execution to find how close you are to a *WS FULL*. Event trapping can help, but it's far from a complete solution.

Until recently, SHARP APL Systems were very isolated. There was no way for an APL function to read (or write) "normal" operating system files. Instead, external data bases had to be stripped, fed into APL in sequential form, and rebuilt in APL files. Since actual operating data, normally processed by batch systems, is an essential input to the decision-makers, this was a very real limitation. Thanks to the shared variable processor, we now have the means to eliminate this problem.

Finally, I am not aware of any practical widely-used general-purpose data base manager in SHARP APL. Each system instead has its own set of files, as did batch systems in the '60's. But the batch people realized that, with systems crossing departmental boundaries and the increasing rate of change, it is essential to use tools which allow systems to share data, while protecting them from undue side-effects, when business requirements dictate changes or additions to the structure of the data base. This need is much greater in systems that are trying to keep pace with a decision-maker's imagination.

Software Tools

What can we do to protect ourselves from the problems of APL while we take advantage of its tremendous power? Apart from badgering the implementers for changes, I believe we can do a lot by taking the trouble to build and use libraries of software tools to help us produce and maintain systems. Software tools are especially potent in APL because APL makes it so convenient to use them. The technique I use to build my library is to always be on the lookout for potentially repetitive and useful tasks. Almost all of the several hundred functions in my library were originally written to serve specific systems. But they were written with general use in mind, or generalized later, then documented and saved, so others could benefit from them. Many have gone through revisions to improve their usefulness. And many have been modified in response to other people's suggestions.

The APL programmer can use software tools in all of the following areas:

- general utilities: functions to handle input in a standard way; output processors (which automatically skip perforations, print page headings and footings, etc.); text manipulators (e.g. to centre headings, or strip leading or trailing blanks); and so on.

- program overlaying package: since the size of the workspace is limited, large systems must be implemented as overlays, either of complete workspaces or packets of functions in the same workspace. A standard overlaying package should be used consistently for these systems.

- programs to help readability: the ubiquitous "IF" function, for example.

- function code analyzers: because of the ease with which an APL function can manipulate the text of another function, many tools are feasible in this area which would be much harder to write in a batch environment. For example: analyze the structure of a complex of functions; return a list of all non-local names referenced by a function and its subordinates. These tools can be life-savers when you have to maintain someone else's code (or even your own, sometimes).

- code generators: functions that, when supplied parameters by the programmer, actually create other standardized functions (e.g. for table maintenance). This concept can be taken to the level where large portions of the code in entire systems can be mechanically generated from specifications and structure definitions entered by the analyst.

The above list, by no means exhaustive, represents a fair amount of work. But the benefits to be gained are very significant.

- programmer productivity is substantially increased because less time is spent re-inventing the wheel

- systems are more easily understood because there is less unique code in them

- standards are easier to promote since the tools that are available encourage or require adherence.

The most important factor, however, in the production of high-quality systems is not technical. It is the environment within which the systems are produced. The APL programmers must work as a cooperative team, not as individuals. The team leader, while respecting each person's unique style and contribution, must encourage free and open discussion and joint attacks on problems. And users must do their part by respecting the advice of the technicians on technical matters though not hesitating to insist that the systems respond to genuine business needs.

For years, APL has suffered many stigmas. Its advocates have been regarded as "APL nuts", or worse. But gradually it seems that APL is gaining respect in the DP world. In some cases the writing is on the wall because users, having seen what APL can do, demand it with or without the support of the DP department. We, in turn, must strive for higher levels of professionalism in our work. It is not enough to develop a system twice, or even ten times, as fast, as we often can in APL. We must also use our tools to give decision-makers the flexibility they need in their use of our systems. And we must be able to react quickly and responsibly when the world turns topsy-turvy on us, either in reality or in the mind of an executive.

APL IN THE CLASSROOM

E.M. (Ted) Edwards
Simon Fraser University
Burnaby, B.C.

Abstract

The author has used APL as a teaching tool in courses directed at students ranging from high school to senior year at university. Topics in these courses have included computer programming, curve fitting, mathematical logic, set theory, predicate calculus, computer hardware design and others. APL has been used as a language of discourse, as a programming tool for student use, and to produce software for editing lesson material and for presenting examples "on-line" in the classroom. Examples of lesson material are included and the reasons for the success of APL in this environment are discussed. Some limitations are also examined and a few possible extensions suggested.

Introduction

The author has used APL in the classroom in two distinctly different modes. In the first, the objective has been to teach APL and in the second, to teach some other topic. In this second case, APL has been used as a language of discourse. That is, having examined various notations used in the subject field, the author felt that APL offered the students advantages that more than offset the overhead of learning a new notation.

Regrettably, many teachers of elementary mathematics courses overlook the power of array notations to convey functional concepts through the examination of patterns (something the human mind is very good at). This oversight occurs since it is generally believed that arrays and array operations are much more difficult to understand. This is, in fact, true if one restricts oneself to conventional notation. APL provides a notation in which the generalization from single element operations to array operations occur in a straightforward and comparatively simple fashion. Some examples will be given below. Fortunately, APL has (to a very large extent) the property that "what you don't know won't hurt you". That is APL may readily be subsetted or, in other words, the teacher need only introduce as much as is needed to meaningfully discuss the subject matter at hand. Further, even this subset may be introduced only as each aspect is needed. In any course the author has every taken or taught, this has occurred anyway as the instructor found it convenient to introduce special symbolisms to expedite the discussion. It is not, then, as drastic a step as it may appear to introduce APL as a notation. In every class the author has taught in which APL has been involved, the concept of scalar functions applied to arrays has been introduced early in the first lecture without apparent difficulty. (See sample lessons included in this paper.)

At the present time, few textbooks are available in which APL has been used as notation. This can mean a considerable amount of extra work for the instructor as it must be recognised that the order of topics and their presentation must often be radically altered. The problem is akin to that of translating poetry. The idioms available for expression are different in different languages. A good translation must present the material in idiomatic form in the target language. In the context of this discussion, that often means considerable effort rearranging the material for presentation in APL. It is the opinion of this author that the investment is often warranted.

Teaching APL

Shown below is the outline for a three week course in computer programming taught by the author. This course is offered as part of the Simon Fraser University Summer Computer Institute. It is directed primarily towards high school students as an enrichment program. Students from grades eight through twelve plus a few of their teachers have taken the course. The primary objective is to learn to program a computer in APL, although the concept of the notational use of APL is introduced early and used throughout. Those students who have some computing background in another language (usually BASIC) present some of the greatest difficulties in terms of misconceptions about computing that need to be dynamited. The worst of these are with regards to modularity and the nature of iteration as distinct from loops used to express what are in reality disguised array operations.

Easy as APL ○1
An Introduction To APL
Course Outline

LESSON 1
- Expressions and results
- Some APL primitive functions
- Vectors
- Names for data objects
- Defining your own functions

LESSON 2
- Tables and matrices
- More primitive functions
- Indexing
- Character strings
- Arrays of truth values
- Graph plotting as an example of array thinking

LESSON 3
- Some more primitive functions
- Automatic scaling for a graph plotter
- Manipulating arrays

LESSON 4
- Character strings and their handling

LESSON 5
- What is inner product?

LESSON 6

Simultaneous equations the easy way

LESSON 7

Polynomials and curve fitting

LESSON 8

An example of visualization of array operations
Projections in N-dimensional spaces

The remaining time will be spent assisting students to implement a project of their own.

The first eight to ten days are spent going through the material of the eight lessons. Mornings are spent in classroom lectures and afternoons are spent gaining "hands-on" experience. An MCM APL computer connected to two large screen video monitors is used to present the material of the lessons in class. A set of hard copy notes is also given to the students. The technique of preparation and presentation of the lesson material is the subject of another paper (Edwards 1). Students have access to the university's MTS APL system running on an IBM 370/148 and to some MCM APL stand-alone systems for their practice sessions. The final week of the course is spent on individual projects with the instructor and an assistant available to provide help and to suggest appropriate programming techniques. Portions of lessons 1, 5 and 8 are shown below as an indication of the presentation.

```
      ⍝                EASY AS APL o1 (LESSON 1)

      ⍝EXPRESSIONS AND RESULTS
      2+3
5
      2+3+4+5
14
      (2+3)×(4+5)
45
      ⍝MAXIMUM ('⌈' IS READ 'MAX')
      2⌈3
3
      7⌈5
7
      ¯3⌈¯1
¯1
      ⍝AS YOU WOULD EXPECT, '⌊' IS MINIMUM.
      2⌊3
2
      ⍝RIGHT TO LEFT EVALUATION ORDER
      2+3×4+5
29
      2+5⌈3+4
9
      9÷2
4.5
      4+6÷2+1
6
```

```
      ⍝VECTORS
      2 3 1+3 4 5
5 7 6
      2 2 2+3 4 5
5 6 7
      2+3 4 5 6
5 6 7 8
      5⌈3 4 5 6 7
5 5 5 6 7
      2 3 4×5 6 7
10 18 28
      2 3 4⌈5 3 2
5 3 4
      2 3 4⌊5 3 1
2 3 1
      3⌊1 2 3 4 5
1 2 3 3 3
      ⍝
      ⍝'⍝'IS READ LAMP.  THE STATEMENT THAT FOLLOWS IT
      ⍝IS FOR ILLUMINATION ONLY (I.E. A COMMENT).
      ⍝NAMES
      DATA←2 3 4 5 6
      ⍝NAME: A STRING OF LETTERS OR NUMBERS BEGINNING WITH A
      ⍝LETTER.  '←' IS READ 'IS' AND ASSIGNS A VALUE TO A NAME.
      DATA
2 3 4 5 6
      2×DATA
4 6 8 10 12

      ⍝               EASY AS APL ○1 (LESSON 5)
      ⍝
      ⍝       INNER PRODUCT
      ⍝
      ⍝AN EXAMPLE OF AN INNER PRODUCT EXPRESSION:
      2 3 4+.×3 2 1
16
      ⍝OBSERVE THAT WE OBTAINED A ONE ELEMENT RESULT FROM TWO
      ⍝VECTOR ARGUMENTS.  YOU SHOULD VERIFY THAT IT WAS IN FACT A
      ⍝SCALAR.  CONSIDER THE TWO EXPRESSIONS:
      2 3 4×3 2 1
6 6 4
      +/2 3 4×3 2 1
16
      ⍝FOR VECTORS α AND ω, AND SCALAR DYADIC FUNCTIONS F AND G,
      ⍝(αF.Gω)=F/αGω.  CURRENT APL'S ALLOW INNER PRODUCTS TO BE
      ⍝FORMED FROM PRIMITIVE FUNCTIONS ONLY.  INNER PRODUCTS MAY
      ⍝BE APPLIED TO HIGHER RANK ARRAYS BY REGARDING THEM AS
      ⍝ARRAYS OF VECTORS.  STUDY THE EXAMPLES BELOW TO SEE HOW
      ⍝THIS IS DONE.

      ⍝                      2  5  4  3  1
      ⍝     +.×              3  5  4  2  2      B
      ⍝                      5  4  3  1  3
      ⍝                      5  4  2  2  5
      ⍝     A
      ⍝ 1  2  3  1          28 31 23 12 19
      ⍝ 2  3  1  2          28 37 27 17 21      A+.×B
      ⍝ 3  1  2  3          34 40 28 19 26
      ⍝
      +/ 2 3 1 2 × 4 4 3 2
```

```
27
      ⍝+.× IS 'ORDINARY' MATRIX MULTIPLICATION
      ⍝               EASY AS APL ○1 (LESSON 8)
      ⍝
      ⍝     PROJECTIONS
      ⍝
      ⍝SAY WE HAVE A VECTOR IN 3-SPACE
      L←1 2 3
      ⍝WE WOULD LIKE TO FIND THE PROJECTION OF L ONTO THE X-Y
      ⍝PLANE I.E. THE PLANE SPANNED BY
      M←1 0 0
      N←0 1 0
      ⍝WE COULD PUT THESE TWO VECTORS INTO A MATRIX
      ▼O←⍉2 3⍴M,N
1 0
0 1
0 0
      ⍝ANY VECTOR IN THE PLANE SPANNED BY THE COLUMNS OF O CAN BE
      ⍝OBTAINED BY (O[;1]×P[1])+O[;2]×P[2] I.E. O+.×P FOR SOME 2
      ⍝ELEMENT VECTOR P.  THUS WE CAN SPEAK OF THE PLANE, O.  IF
      ⍝L LAY IN THE PLANE O, THEN WE COULD FIND P BY SOLVING
      ⍝L=O+.×P.  SIMULTANEOUS EQUATIONS AGAIN!  WELL, LET'S TRY
      ⍝SOLVING WITH ⌹ (OR QD∆) EVEN IF L ISN'T IN O.
      ▼P←L QD∆ O
1 2
      ⍝IF WE ADD 1 OF M AND 2 OF N WE GET:
      M+2×N
1 2 0
      ⍝WHICH IS THE DESIRED PROJECTION!  BUT THIS IS JUST:
      (O[;1]×P[1])+O[;2]×P[2]
1 2 0
      ⍝WHICH IS:
      O+.×P
1 2 0
      O+.×L QD∆ O
1 2 0
      ⍝WOULD IT MATTER IF THE COLUMN VECTORS OF O WERE NOT OF
      ⍝LENGTH 1?  LET'S MAKE O[;1]=2 0 0 AND FIND OUT.
```

Defined functions are introduced in lesson 1 and an example is included in which one function calls another. Outer product is dealt with in lesson 2. Direct definition (Iverson 1) (the alpha/omega notation) is used throughout. 'Del" definition is explained near the end of the course. As may be seen, array concepts are introduced early and used throughout. In fact, the students are given no mechanism to create a loop until well on.

Teaching With APL

Some texts have been produced in which APL has been used as the language of discourse. Excellent examples are Blaauw 1, Iverson 1, 2, Orth 1 and Spence 1. Iverson's texts deal with high school algebra and elementary analysis, Orth's deals with calculus and Blaauw's and Spence's with electronics. In the spring semester of 1980, the author was asked to teach a course in discrete mathematics. This course has previously been taught in the conventional manner and covers the material shown in the outline below but in a different order. The author decided that this material was a natural place for the use of APL as notation. As no suitable text was available, the author developed a set of notes for the course as a first step to producing such a text. A sample of some of the material is shown below.

The usual presentations seem to treat the topics of logic, set theory and relations almost as totally separate topics with a different notation for each. Eventually, students spot the connections between these topics, but usually not while they are taking the course. It is this author's opinion that set theory is most readily introduced via the characteristic vector used to indicate the truth or falsity of the proposition "these elements of the universe of discourse belong to this set".

Relations may be introduced by generalising the characteristic vector to a characteristic array with a matrix representing a set of sets. Boolean matrices are a natural way to deal with cartesian product sets, partitions, and other topics where sets of sets are the objects of study.

CMPT 205-3 - FALL 1980
Introduction To Formal Topics in Computing Science

INSTRUCTOR: E.M. Edwards
C.C. 7318
291-3761

This course introduces some of the theoretical tools and concepts used in computing science. The purpose of the course is to provide the student with some facility at recognizing that some problems may be modelled in formal systems, and in using such formal systems to advantage in solving problems and designing algorithms.

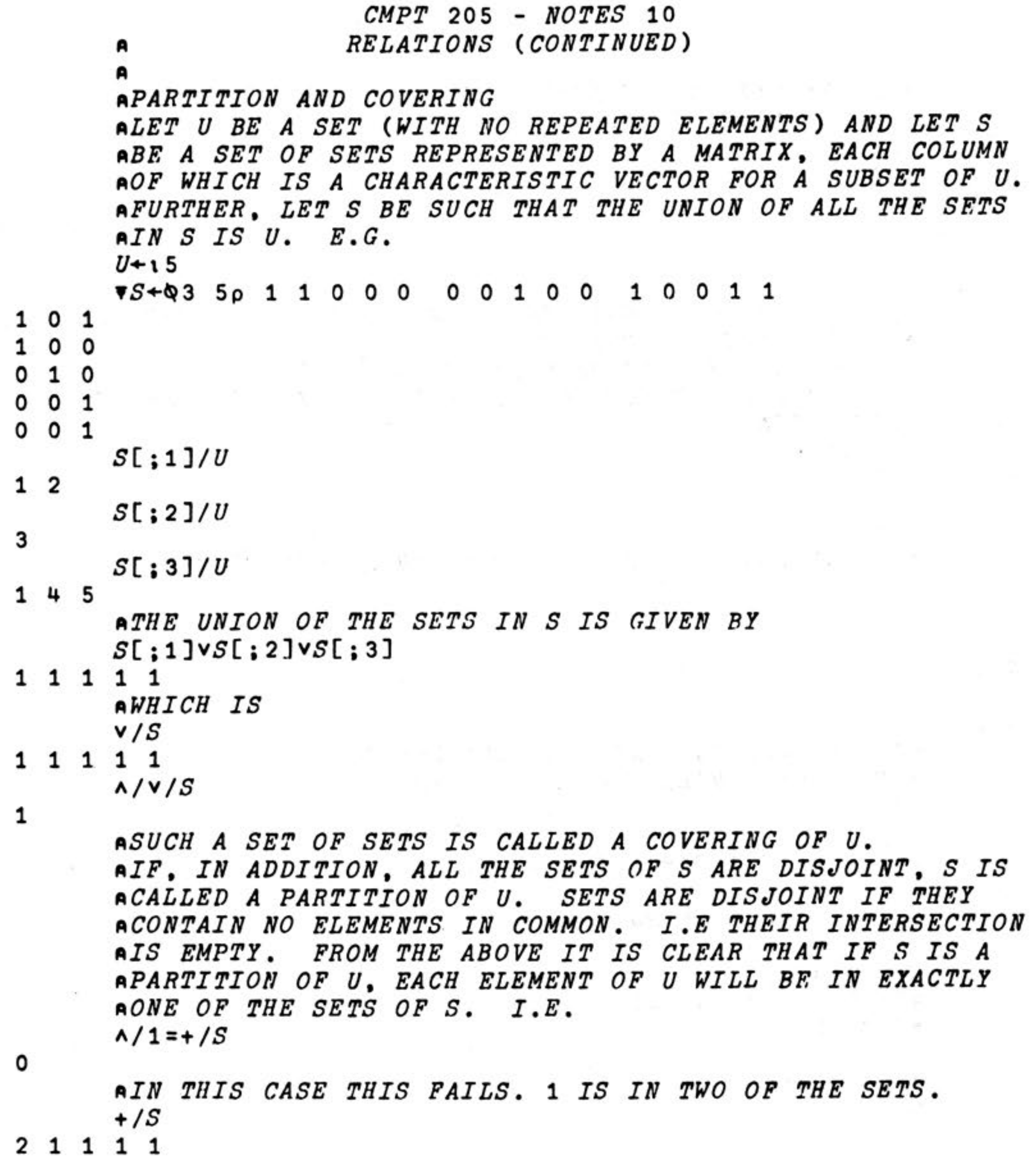
APL is used as notation for developing concepts and proving theorems as well as for programming examples of the theory. The material has been arranged so as to gradually introduce the necessary subset as required. No prior knowledge of APL is assumed.

The main topics to be covered are:

1. Logical statements, truth values and truth tables

2. Logical functions and identities
3. Introduction to set theory
4. Predicate calculus
5. Relations and graphs
6. Equivalence relations and partitions
7. Ordering relations and the basis for sorting
8. Functions
9. Some aspects of number theory

TEXT:
"Computing Science 205 Notes", E.M. Edwards
"Easy As APL ○1", E.M. Edwards



```
                        CMPT 205 - NOTES 10
        ⍝               RELATIONS (CONTINUED)
        ⍝
        ⍝PARTITION AND COVERING
        ⍝LET U BE A SET (WITH NO REPEATED ELEMENTS) AND LET S
        ⍝BE A SET OF SETS REPRESENTED BY A MATRIX, EACH COLUMN
        ⍝OF WHICH IS A CHARACTERISTIC VECTOR FOR A SUBSET OF U.
        ⍝FURTHER, LET S BE SUCH THAT THE UNION OF ALL THE SETS
        ⍝IN S IS U.  E.G.
        U←⍳5
        ▼S←⍉3 5⍴ 1 1 0 0 0  0 0 1 0 0  1 0 0 1 1
1 0 1
1 0 0
0 1 0
0 0 1
0 0 1
        S[;1]/U
1 2
        S[;2]/U
3
        S[;3]/U
1 4 5
        ⍝THE UNION OF THE SETS IN S IS GIVEN BY
        S[;1]∨S[;2]∨S[;3]
1 1 1 1 1
        ⍝WHICH IS
        ∨/S
1 1 1 1 1
        ∧/∨/S
1
        ⍝SUCH A SET OF SETS IS CALLED A COVERING OF U.
        ⍝IF, IN ADDITION, ALL THE SETS OF S ARE DISJOINT, S IS
        ⍝CALLED A PARTITION OF U.  SETS ARE DISJOINT IF THEY
        ⍝CONTAIN NO ELEMENTS IN COMMON.  I.E THEIR INTERSECTION
        ⍝IS EMPTY.  FROM THE ABOVE IT IS CLEAR THAT IF S IS A
        ⍝PARTITION OF U, EACH ELEMENT OF U WILL BE IN EXACTLY
        ⍝ONE OF THE SETS OF S.  I.E.
        ∧/1=+/S
0
        ⍝IN THIS CASE THIS FAILS. 1 IS IN TWO OF THE SETS.
        +/S
2 1 1 1 1
```

```
⍝IF WE MODIFY S SLIGHTLY, IT BECOMES A PARTITION
S[1;3]←0
▼S
1 0 0
1 0 0
0 1 0
0 0 1
0 0 1

      ∧/1=+/S
1
      ⍝S IS A PARTITION SINCE ∧/1=+/S.  THE ELEMENTS OF A
      ⍝PARTITION ARE CALLED BLOCKS.  S HAS THREE BLOCKS.
      ⍝OTHER PARTITIONS OF U ARE POSSIBLE.  E.G.
      5 1⍴1
1
1
1
1
1
      ⍝THE PARTITION CONSISTING OF ONE BLOCK CONTAINING ALL
      ⍝THE ELEMENTS OF U.
      (⍳5)∘.=⍳5
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
      ⍝THE PARTITION CONSISTING OF ALL SETS WHICH CONTAIN
      ⍝EXACTLY ONE ELEMENT OF U.
      ⍝LET B BE A SUBSET OF U REPRESENTED BY ITS
      ⍝CHARACTERISTIC VECTOR.  THEN B AND ITS COMPLEMENT
      ⍝PARTITION U.
      ▼B←U∊2 4
0 1 0 1 0
      ~B
1 0 1 0 1
      ⍝WE MAY CONVENIENTLY GENERATE THIS PARTITION BY
      ▼P←B∘.=1 0
0 1
1 0
0 1
1 0
0 1
      ⍝TO SEE WHY THIS WORKS, CONSIDER THAT B=1 LEAVES B
      ⍝UNCHANGED WHILE B=0 IS THE SAME AS ~B.
      ∧/1=+/P
1
      ⍝THUS P IS A PARTITION OF U.
      ⍝
      ⍝LET B AND C BE SUBSETS OF U.
      B←U∊3 4
      C←U∊2 4 5
      ⍝THEN THE FOUR SETS
      ▼I0←(~B)∧~C
1 0 0 0 0
      ▼I1←(~B)∧C
0 1 0 0 1
```

```
      ▼I2←B∧~C
0 0 1 0 0
      ▼I3←B∧C
0 0 0 1 0
      ⍝FORM A PARTITION OF U.
      ▼I←⍉4 5⍴I0,I1,I2,I3
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
0 1 0 0
      ∧/1=+/I
1
      ⍝AS CAN BE SEEN, I IS A PARTITION OF U.  THE SET I IS
      ⍝CALLED THE COMPLETE INTERSECTIONS OR THE MINTERMS
      ⍝GENERATED BY B AND C.  NOTE THAT THE POSITIONS OF THE
      ⍝'~' SYMBOLS IN THE EXPRESIONS FOR I0, I1, I2 AND I3
      ⍝CORRESPOND TO THE 0'S IN THE 2 DIGIT BINARY
      ⍝REPRESENTATION OF 0, 1, 2 AND 3.  WE MAY EXTEND THIS
      ⍝TO N SETS AND ELIMINATE EMPTY SETS AS FOLLOWS:
      MINTERMS:(∨⌿R)/R←⍵∧.=BITS ¯1↑⍴⍵
MINTERMS
      BITS:(⍵⍴2)⊤¯1+⍳2*⍵
BITS
      ⍝TO SEE THIS, CONSIDER THAT ∧/ IS THE INTERSECTION OF A
      ⍝SET OF SETS AND =1 OR =0 GIVES THE SET OR ITS
      ⍝COMPLEMENT (AS SHOWN PREVIOUSLY).  IF THIS IS APPLIED
      ⍝TO THE INNER PRODUCT DIAGRAM, THE RESULT IS CLEAR.
      ⍝REMEMBER THAT (N⍴2)⊤¯1+⍳2*N GIVES THE BINARY NUMBERS
      ⍝FROM 0 TO ¯1+2*N AS ITS COLUMNS.
      U←⍳10
      ▼S←⍉3 10⍴(U∊1 2 5 6 9),(U∊2 3 4 5 9),U∊4 5 6 7
1 0 0
1 1 0
0 1 0
0 1 1
1 1 1
1 0 1
0 0 1
0 0 0
1 1 0
0 0 0


      ▼MS←MINTERMS S
0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0
      ∧/1=+/MS
1
      ⍝SO MS IS A PARTITION OF U.
      ⍝EMPTY SETS MAY BE ADDED TO OR REMOVED FROM A PARTITION
      ⍝WITHOUT AFFECTING ITS BEING A PARTITION SINCE AN EMPTY
      ⍝SET IS DISJOINT FROM EVERY SET AND CONTAINS NO ELEMENTS.
```

Below is a sample of a proof using APL notation. Since the notation is machine executable, an example is run in parallel with the steps of the proof to provide a partial check against errors (e.g. a wrong sign).

```
      ⍝THEOREM: (EQUIV R) ≤ (~R[A;B]) = ∧/~R[A;]∧R[B;]
      ⍝IN WORDS: LET R BE AN EQUIVALENCE RELATION.
      ⍝THEN A AND B ARE NOT RELATED IFF THE R-RELATIVES
      ⍝OF A AND THE R-RELATIVES OF B ARE DISJOINT.
      A←2∘B←5
      ⍝PROOF THAT (∧/~R[A;]∧R[B;]) ≤ ~R[A;B]
      (∧/~R[A;]∧R[B;]) ≤ ~R[A;B]∧R[B;B]      ⍝INSTANCE
1
      R[B;B]                                   ⍝REFLEXIVE
1
      (∧/~R[A;]∧R[B;]) ≤ ~R[A;B]               ⍝A=A∧1
1
      ⍝PROOF THAT (~R[A;B]) ≤ ∧/~R[A;]∧R[B;]
      P←∧/~R[A;]∧R[B;]
      P = ~∨/R[A;]∧R[B;]           ⍝DE MORGAN
1
      P = ~∨/R[A;]∧R[;B]           ⍝SYMMETRY
1
      (~P) = ∨/R[A;]∧R[;B]         ⍝(P=Q) = (~P)=~Q
1
      (~P) ≤ R[A;B]                ⍝TRANSITIVE
1
      (~R[A;B]) ≤ P                ⍝(P≤Q) = (~Q)≤~P
1
      ⍝QED
```

Some Extensions to APL

Some of these (or similar) suggestions have appeared elsewhere. They are included here since their absence has been particularly noticed in classroom applications.

1. Allow the comment symbol to appear anywhere on a line. The processor would ignore everything after its first appearance.

2. Specification into selection expressions. An extremely potent feature of APL is the facility with which selection of sub-arrays may be done. Unfortunately, this aspect of current implementations is not symmetric with respect to "read" and "write" operations. It would be desirable to replace elements of an array with the same facility that allows them to be tested or examined.
   e.g.

```
      ⍝INSERT TWO 1'S ON THE DIAGONAL OF A 2×2 MATRIX SITUATED
      ⍝WITH ITS UPPER LEFT CORNER AT M[2;3] WITHOUT CHANGING ANY
      ⍝OTHER VALUES.
      M←4 6⍴0
      (1 1⍉2 2↑1 2↓M)←1
      M
0 0 0 0 0 0
0 0 1 0 0 0
0 0 0 1 0 0
0 0 0 0 0 0

      ⍝GIVEN EQUAL LENGTH VECTORS OF ROW AND COLUMN INDICIES, I
      ⍝AND J, INSERT VALUES X IN POSITIONS OF M GIVEN BY
      ⍝M[I[1];J[1]],M[I[2];J[2]],...
      (1 1⍉M[I;J])←X

      ⍝INSERT 1'S IN M[1;2], M[2;4], M[3;6], ...
      ((¯1+2×⍳1↑⍴M)⌽M)[;1]←1
```

3. Extension of operators to defined and composite functions. The principle of syntactical equivalence (i.e. defined functions and primitive functions are syntactically equivalent) currently fails for operators since defined or composite functions may not be indexed nor may they be used as arguments for reduction, scan and inner and outer product. The major difficulty to be overcome in proposing this extension is the handling of identities for reduction of empty arrays.

```
      ⍝THE SYMBOL '¨' WILL BE USED IN THE DEFINITION FOR A
      ⍝FUNCTION INDEX.
      AVG:(+/[¨]⍵)÷(⍴⍵)[¨]
      ▼X←3 4⍴⍳12
1  2  3  4
5  6  7  8
9 10 11 12
      AVG X
2.5 6.5 11.5
      AVG[1] X
5 6 7 8
```

```
⍝IF A RELATION IS REPRESENTED BY A SQUARE BOOLEAN MATRIX,
⍝R, THE TRANSITIVE CLOSURE, S, OF R IS GIVEN BY:
S←∨.∧/[⎕IO](3⍴⍴R)⍴R

⍝MONTHLY IS A DEFINED SCALAR DYADIC FUNCTION WHICH RETURNS
⍝THE MONTHLY PAYMENT REQUIRED TO AMORTISE A $1 LOAN OF
⍝DURATION ⍵ AND MONTHLY INTEREST FACTOR ⍺.  PRODUCE A
⍝TABLE OF MONTHLY PAYMENTS FOR A $1200 LOAN AT MONTHLY
⍝INTEREST RATES OF 1, 1.5, AND 2 PERCENT WITH DURATIONS OF
⍝1, 2, 3, AND 5 YEARS.
TABLE←.01×⌈100×1200×.01 .015 .02∘.MONTHLY 12×1 2 3 5
```

4. Extended scalar conformability. Scalar dyadic functions should accept arguments of equal rank provided that, element by element, either their dimensions are equal or at least one of the pair of values is 1. In the event the ranks of the arguments differ by 1, the dimension to be coerced is supplied by indexing the function with the usual default of last dimension. This would permit such operations as:

```
⍝ADD ⍳5 TO EACH ROW OF A 5×5 MATRIX, X.
X←X+[⎕IO]⍳5

⍝MANY OF THE USES OF THIS EXTENSION LIE IN COMBINATIONS
⍝OF A REDUCTION AND A SCALAR FUNCTION.  E.G.

⍝SCALE THE ROWS OF A MATRIX SO THAT EACH LIES IN THE RANGE
⍝0 TO 1.
SX←SX÷S+0=S←⌈/SX←X-⌊/X

⍝FORCE THE MEAN OF EACH COLUMN OF X TO 0 (SEE AVG ABOVE).
X←X-[⎕IO]AVG[⎕IO]X
```

Conclusion

Materials have been presented from two courses in which APL plays a role, either as the subject being taught or as notation to aid in the discussion of another subject.

Many areas of Mathematics and Physics could benefit from the use of APL as a teaching notation if well planned and executed teaching materials are made available. Maximum benefit is gained when the required APL sophistication is developed gradually along with the subject matter. An APL system should be available to students for examples and problems. Problem sets should mix exercises which require notational and computational use of APL. The advantages of APL are completely lost if use is not made of array concepts such as outer and inner product, reduction, etc. Attempts to transliterate algorithms from other languages on the grounds that "loops are easier to understand" completely miss the point.

References

1. Blaauw, G.A., **Digital System Implementaton**, Prentice-Hall, 1976.
2. Edwards, E.M., **ECHO — A Smart Electronic Blackboard**, 3rd Canadian Symposium on Instructional Technology, 1980.
3. Iverson, K.E., **Elementary Analysis**, APL Press, 1976.
4. Iverson, K.E., **Algebra — An Algorithmic Treatment**, APL Press, 1972.
5. Orth, D.L., **Calculus in a New Key**, APL Press, 1976.
6. Spence, R., **Resistive Circuit Theory**, McGraw Hill, 1974.

AN INTRODUCTION TO SUPERPLOT

Brian W. Oliver
I.P. Sharp Associates Limited
Toronto, Canada

Introduction

Interest in computer graphics developed at I.P. Sharp Associates in the early seventies, with the advent of two new terminal display technologies: the daisy wheel impact printing mechanism, and the TEKTRONIX family of cathode ray graphics screens. Certainly, computer graphics devices existed before this time, but they were extremely expensive, and not readily adaptable to a timesharing protocol. Since it was not known in the beginning what sorts of graphics capabilities would be required by the user community, the product evolved in sophistication as user experience provided valuable feedback. One distinction was apparent early: separate capabilities were desirable for object graphics (general shapes, contour maps, cross sections, etc.) and plotting (axis-oriented). This paper deals with the development and current status of the latter product: SUPERPLOT.

What is SUPERPLOT?

SUPERPLOT first appeared in 1976 in MAGIC (see **Data Manipulation with MAGIC** [1]) as a data display alternative to the *DISPLAY* and *TABLE* commands. A plot command, *PLOT*, had previously existed (and still does) but it produced only "point plots" which were often difficult to interpret, as Figures 1 and 2 indicate. The new daisy wheel terminals offered variable carriage spacing both horizontally and vertically, and made point interpolation possible. Thus, beginning in 1976, MAGIC users were able to produce plots with solid lines instead of data points. Figure 3 demonstrates the advantage of a line plot, with the same data used in Figure 2.

Over the next four years, with continuous assistance from MAGIC users, SUPERPLOT was enhanced with many new features and capabilities, and became an extremely versatile software package, capable of producing almost any type of data plot previously done manually. During this period, the major design features were:

1. Device Independence
2. Conversational/Nonconversational Modes
3. Output Versatility
4. Timeseries Independence
5. Geographic Independence

and a short discussion of each feature follows.

1. Device Independence

Although most graphics terminal devices fall into three categories (daisy wheel impact, graphics screen, flatbed plotter), each manufacturer's model contains internal differences that make it necessary for SUPERPLOT to be aware of the device model in order to issue the correct terminal movement instructions. As new models are introduced by the manufacturer, the graphics support group at I.P. Sharp obtains the technical specifications for the machine, and modifies SUPERPLOT to recognize the model. At present, the current terminal devices recognized and supported by SUPERPLOT are:

Daisy Wheel Impact	**Graphics Screen**	**Flatbed Plotter**
Xerox Diablo 1550, 1620	TEKTRONIX 4013	Hewlett Packard 7221
1641, 1650, 1700, 1740	TEKTRONIX 4015	TEKTRONIX 4662
1750		Calcomp 81
Trendata 4000, 4000A		
Anderson Jacobson 832		
DTC 300, 302		
ASL2, ASL3		

2. Conversational/Nonconversational Models

Because of the sophistication of SUPERPLOT, it was decided in the beginning to depart from tradition in MAGIC, and allow plot preparations to be made conversationally, through user-SUPERPLOT terminal dialogue. This was especially useful to novice users for self-instruction, and for experienced users for plot style experimentation. The nonconversational mode permits insertion of SUPERPLOT commands within a user-defined program, as is desirable in "production systems". Both modes offer the same capabilities and both are often employed by the same user at different stages of system development.

3. Output Versatility

Whereas the variety in which data may be displayed in tabular form is limited, the possibilities for graphical display are infinite. SUPERPLOT users often wish to automate the production of graphs previously drawn by hand, and retain their original formats, which can differ either by industry convention or individual taste. While securities analysts use "high-low-close" formats, economists may prefer histograms and bar charts and aviation industry professionals might have a fondness for line plots. To be as generally applicable as possible, SUPERPLOT had to make all these formats, and many others, readily available. Currently, the **major** plot characteristics over which the user has control are:

SIZE - the physical dimensions of the plotted output

LINE TYPE - histogram, line, high-low-close, etc.

LINE STYLE - solid, dotted, dashes, etc.

LABELLING - of the title, legend, axes

COLOURS - if the device is a flatbed plotter

Several other format "options" exist, mostly as refinements of the five major features above. Recommendations and suggestions for new format features are earnestly welcomed.

4. Timeseries Independence

In early 1980, a series of new commands relating to the new *NOTIMESERIES* feature, were announced in MAGIC. Naturally, SUPERPLOT had to be modified to permit non-timeseries plots, with neither axis being time oriented. This elimination of reliance on time as one dimension of data made MAGIC much more generally applicable as a data management system, and SUPERPLOT as a generalized plotting facility. Figure 4 illustrates a capability that was not possible in SUPERPLOT prior to 1980: a graph where periodicity is not inherent in the display of data. With this new capability, it was decided that SUPERPLOT was powerful and general enough to merit its own public workspace, independent of MAGIC.

5. Geographic Independence

Although SUPERPLOT is device independent, it still requires a terminal with graphics features, and not every user of SHARP APL has such a machine. A recent enhancement to SUPERPLOT will allow a user with **any** model of timesharing terminal to specify the format and data for a plot and then have it produced elsewhere (an I.P. Sharp branch office) with instructions for delivery of the output. This "GRAPHREQ" feature was designed to be consistent with the SHARP APL highspeed printing (HREQ) and B-task submission (BTASKREQ) facilities. Remote plot output naturally involves a time delay but offers any SHARP APL user access to virtually a complete range of plotting machinery.

Where is SUPERPLOT?

For users with no need for MAGIC, SUPERPLOT exists on a "stand alone" basis in the public workspace 3 *SUPERPLOT*. As well, SUPERPLOT is an integral part of MAGIC and is available in any of the public data base workspaces containing MAGIC:

```
 39 MAGIC          702 INS      702 BAREMAGIC
121 PETROSERIES    702 ER586
121 IMPORTS        702 T6
702 MAGIC          702 ICAO
```

Each of the workspaces above contain the current versions of MAGIC and SUPERPLOT. Renaming any of these and storing them as a workspace in a private library could cause unnecessary storage expense and run the risk of becoming obsolete; enhancements to MAGIC and SUPERPLOT may not always be reflected in private versions.

Using SUPERPLOT

There are two separate steps involved in producing plot output. The first step is the specification of the format attributes (options) of the plot, using the command *SUPERPLOT*. The second step is drawing of the output to the specifications, using the *PLOT* command. Once options have been set, they remain in effect through successive plot productions until respecified.

Setting Plot Options

When used in the conversational mode, *SUPERPLOT* operates a multi-level dialogue structure. The primary level prompts the user for an option keyword with the standard prompt:

OPTION:

and there are four possible responses to this primary prompt:

1. *STOP* - causes termination of the option setting phase and exits the *SUPERPLOT* program

2. carriage return - causes the printing of a list of all possible responses to the prompt

3. *HELP* - causes the printing of a detailed description of each possible response

4. An option keyword; one of:

AXES
CHARS
COLOUR
DECIMALS
DIVIDER
EDGE
FILE
FONT
FOOTNOTE
GRAPHREQ
GRID
GROUP
HIGHLIGHT
LABEL
LEGEND
LINE
MULTI
OBSERVATION
OFFSET
ORDERED
PASSES
POSITION
RANGE
RESET
SHADING
SHOW
SIZE
STAMP
STYLE
TERMINAL
TITLE
TYPE
UNITS
XAXIS
XLABEL
XSCALE
XTEXT
XTIK
YAXIS
YLABEL
YSCALE
YTEXT
YTIK
ZERO

Specifying an option keyword moves the dialogue to the secondary level in which particulars of the chosen option must be provided. The prompts at this secondary level are specific to the particular option and the *STOP*, *CR* (carriage return) and *HELP* responses have the same effect as in the primary level. The *TERMINAL* option, which identifies the output device which will actually draw the graph is the only option which **must always** be specified; all other options have "default" values which are used if the option is not set by the user. Figure 5 shows an example of minimal option designation in *SUPERPLOT*, where only the terminal type and Y axis labels are set. In this and all subsequent examples, the *PLOT* command is issued to produce an output sample.

The *SIZE* option determines the physical dimensions of the resultant graph. The example in Figure 6 illustrates the production of a plot intended to fit a letter-size 8½ by 11 inch page. Note, in this figure that option names and their prompt responses may be specified on a single line (with comma separators) and abbreviated.

Figure 7 is an example of usage of the the *STYLE* option. This option determines the line style (solid, dotted, dashed, etc.) which is useful for making the lines on a crowded plot distinguishable. All previously described options are used in this example, and the *PLOT* command is used with a specification to its left. Each letter in this left argument has a separate effect:

L - print the left side Y axis
R - print the right side Y axis
T - print the top X axis
B - print the bottom X axis
H - print a highlight box around the plot
G - print a grid within the axes

When no left argument is provided to *PLOT*, the default axes are printed: left and bottom, as illustrated by the earlier examples. Also in Figure 7, the *CHARS* option was used to specify, for each line plotted, which character was to be printed at the observation points (data points). In the currency exchange rates example, a carriage return by itself was supplied to the *CHARS* prompts, indicating that **no** plot character was to be printed at the observation points.

The examples presented up to this point illustrate various display alternatives, but all of them deal with plotted observations joined by straight lines. Quite often, manually drawn graphs contain not straight lines but curved lines, histograms, bar charts, multibars and the high-low-close format for displaying trading prices. The example in Figure 8 uses the *TYPE* option of SUPERPLOT, which allows individual specification of each line as either a straight line (default type) or in one of the formats described above. In this example, the plot legend (established with MAGIC's *LABEL* command) was drawn below the graph. This occurred because SUPERPLOT was unable to find enough blank space within the axes to locate the legend. The *LEGEND* option in SUPERPLOT allows the user to force the position of the legend; if this option is not specified, the legend "floats" to the most appropriate free space.

The *TYPE*, *STYLE* and *CHARS* options all relate to the format and nature of the lines to be drawn by SUPERPLOT. Another option, *LINE*, logically incorporates these and a few other options to permit complete specification of any individual line or group of lines, as figure 9 shows. In this example, the first three lines are related in a high-low-close line format, and the *LINE* option allows grouping. The fourth line, Share Volume, is alone in a group and can therefore have its own set of attributes. The *LINE* option also allows designation of the *PASSES* option which instructs SUPERPLOT as to how many times the line (or group of lines) is to be redrawn (for thickness), and the *COLOUR* option, which assigns colours to the lines but is only meaningful on a flatbed plotter or a daisy-wheel terminal with bicoloured ribbons. Another new option, *GROUP*, is introduced in this example. It allows lines to be grouped in separate graphs sharing a common X-axis ("split plots"). This is useful when one group of lines uses different Y-axis units (share price versus trading volume) than another, or when the units are the same but the magnitudes are considerably different. The *GROUP* option requires the grouping of lines by number (as does the *LINE* option), and an allocation of percentage of the total physical plot area to each group.

The final sample plot of this section, Figure 10, demonstrates another line type: stacked histogram, which is set by *SHI*. Manually produced graphs with this line type are usually coloured or shaded to enhance readability, and this is accomplished in the example with the *SHADING* option. This option permits the specification of parallel shading lines to be drawn within particular lines (histogram boxes) at certain angles of inclination, colour, and density. Similarly, the *FONT* option serves to enhance graph attractiveness by providing for label characters (title, legend, axis text) to be larger than normal character size. *FONT* is not currently applicable to plots on daisy wheel graphics terminals.

Two more SUPERPLOT options deserve mention, although they do not directly affect the final output: *SHOW* and *RESET*. As described previously, options retain their default values until explicitly set, and retain their specified settings until respecified. This applies to successive plots performed in the same workspace during the same terminal session. The *SHOW* option displays the current settings of designated options, and the *RESET* option returns named options to their default values. Both *SHOW* and *RESET* can operate on a single option, a group of options (separated by commas), or *ALL* options, as shown in Figure 11.

The examples in this section describe only a subset of the options available in SUPERPLOT; a complete description of every option can be found in the SUPERPLOT User's Manual, or through the *HELP* response to any prompt. Those options illustrated however, are the ones most commonly used, and should serve to introduce the nature and structure of option setting within SUPERPLOT.

Plotting Non-Timeseries Data

All previous examples dealt with timeseries data, and all of the sample graphs had X-axes which designated the relevant periods. When the dimension of time is not a consideration in the **display** of data, MAGIC addresses this requirement with the *NOTIMESERIES* facility: the example in Figure 12 illustrates how SUPERPLOT handles data retrieved in MAGIC under *NOTIMESERIES*. In the example, the time dimension is replaced by a series of values (via *COLWISE*) which represents points in the line. The plot is transposed to make the X-axis vertical, by use of the letter *S* in the left argument of *PLOT* ("sideways" plot).

In the next example, Figure 13, the *VERSUS* command is used in conjunction with *PLOT* to make each axis represent a range of values of two separate lines, or groups of lines. In this figure, the Y-axis represents 3 series and is logarithmic in scale, as set by the *YSCALE* option. The X-axis represents a single series and its axis label is set by the *XLABEL*.

Disposal of Plot Output

Typically, a plot is produced immediately after the relevant data has been collected and the SUPERPLOT options have been set, as in all of the previous examples. Two alternatives to immediate production of output have recently been added to SUPERPLOT. The *FILE* option allows plot output to be directed to a specified APL file instead of a terminal. This option permits plot specification by users who are either not using a graphics terminal (all preparations up to the plot printing can be performed on **any** terminal), or do not wish immediate output. Plot output "filed" in this manner may be retained as long as desired, at minimal storage charge. The example in Figure 14 demonstrates the use of *FILE*.

GRAPHREQ is a complementary option to *FILE*. It allows a SUPERPLOT user to produce a graph previously filed, either at the terminal signed on at the time, or at a designated remote location. Accordingly, *GRAPHREQ* is most useful to users with their own graphics machinery who wish to reproduce filed plots (at considerable savings over repetitive use of *SUPERPLOT* and *PLOT* in the conventional fashion), and users who require coloured graphs, but do not have access to a flatbed plotter. Figure 15 is an example of use of *GRAPHREQ* to produce the plot filed in the example in Figure 14. Figure 16 illustrates the submission of a request for remote plot production.

Nonconversational Use of SUPERPLOT

The conversational mode of SUPERPLOT is most useful for graphics experimentation, development, and self-instruction. Plots that are part of a production system are less likely to change their format from one run to the next, and only the data itself varies. Accordingly, in a production environment, user-program dialogue is not as critical and can be time-consuming and needlessly expensive. The nonconversational mode of SUPERPLOT is available through the program *ΔSUPERPLOT*, and permits the setting of plot options within a user-defined program. *ΔSUPERPLOT* is a monadic APL function whose right argument is a character string containing the option settings. Since the *PLOT* function, which actually produces the final output, is always nonconversational, it too can be part of a user's program. Option keywords and secondary level responses supplied to *ΔSUPERPLOT* are the same as used conversationally. Figure 17 is an example of a user-defined program which uses SUPERPLOT nonconversationally. Note that a slash (/) is used to separate options. In the case of options which require multiple responses (e.g. *LINE*), a slash has the effect of ending a specific option and retaining the default values for any response not provided. For example, the expression:

```
      ΔSUPERPLOT 'LINE,1,BAR/'
```

sets the line option which controls line type, style, colour, plot characters, and number of passes. Since only the line number and type are specified, the style, colour, character and passes for line 1 retain their default settings. As shown in Figure 17, *ΔSUPERPLOT* may be used more than once and its effect is cumulative.

Technical Overview of SUPERPLOT

The software which supports the three main functions of the SUPERPLOT package, *SUPERPLOT*, *ΔSUPERPLOT*, and *PLOT*, would more than fill a clear workspace. The solution to the space problem (as opposed to larger workspaces) is an extension of the basic software storage design of MAGIC, which itself also embodies hundreds of thousands of bytes of source code. Both MAGIC and SUPERPLOT exist in the public workspaces as sets of "cover functions": very small (one or two lines) functions which in turn bring the larger, complete function each represents, into the workspace, from an APL file where all supporting software is kept. Cover functions themselves consume minimal space and the net effect of their usage is that only those which are used bring in other functions, and workspace size is kept manageable. An example of a cover function is MAGIC's *TABLE* command. It is a 2 line program which consumes 160 bytes of workspace. If called by the user, it defines another function, *T̲ABLE*, in the workspace (from storage on file), which has 39 lines of code and weighs 4720 bytes. It is this second function which performs the familiar reporting work, and if *TABLE* is not called (in favour of *DISPLAY* perhaps), *T̲ABLE* is not brought in. This technique applies to both MAGIC and SUPERPLOT, and has the added benefit of allowing each of the several public copies to draw from the same source, guaranteeing consistency and simplifying source code maintenance.

SUPERPLOT has its own source code file which is divided into three logical sections. The first section contains "state setting" functions which are invoked during use of *SUPERPLOT* and *ΔSUPERPLOT*, and are mainly concerned with setting options. The second section of the source file is concerned with "plot type dependent" functions, portions of which deal with the various types of plots (line, bar chart, histogram, etc.). The final section contains a set of programs known as "OUTTERMS". One OUTTERM function exists for each model of graphics terminal; that is, for each valid response to the *TERMINAL* option. The OUTTERM is responsible for translating general plot instructions into commands recognizable to the specific device. This modularity allows a new graphics terminal to be supported by SUPERPLOT merely by the addition of a new OUTTERM to the SUPERPLOT source code file.

Typically, SUPERPLOT is accessed by users signed on to SHARP APL as a T-task (terminal connected). Internally, *SUPERPLOT* and *∆SUPERPLOT* both operate within this T-task, as they are primarily responsible for user dialogue and state setting (assignments of values to global variables). *PLOT* however, is largely computational, as it produces the commands which draw the output, through the appropriate OUTTERM. Because of its high computation factor and absence of user dialogue, *PLOT* does most of its work in an N-task (background processing; no terminal attached) which it creates. The N-task accepts the plot specifications from the T-task part of *PLOT*, creates the plot commands, and passes the final set of instructions (a *⎕ARBOUT* string) back to the T-task, which is responsible for driving the terminal. Communications between the T-task and N-task segments of *PLOT* are done with shared variables and buffering, to prevent possible *WSFULL* conditions. By employing an N-task to assume the CPU burden, the resultant timesharing costs of producing a plot are appreciably reduced. Users wishing to measure the cost of a plot can get T-task resource consumption from *⎕AI*, and the N-task CPU units consumed from the global variable *∆AICPU*, which is created in the workspace upon completion of the N-task.

Error handling by SUPERPLOT is extensive, through advanced use of *⎕TRAP*. Errors encountered during execution of *SUPERPLOT* will result in corrective dialogue with the user. Errors in the nonconversational *∆SUPERPLOT* and *PLOT* cause display of an appropriate message to the terminal (T-task), and if the error cannot be corrected, the program is abnormally terminated. In every case of program error, the relevant *⎕ER* and other helpful diagnostic information is appended to an "error log" file for examination by the graphics support group at I.P. Sharp.

SUPERPLOT Documentation

SUPERPLOT documentation exists in two forms: on-line and printed. The on-line documentation is available through the variable *DESCRIBE* in public workspace 3 *SUPERPLOT*, and through the *HELP* facility in the conversational *SUPERPLOT*. Printed documentation can be found in both the MAGIC User's Manual, and the SUPERPLOT User's Manual.

New features, or enhancements to existing features, made subsequent to the publication of printed documentation are described in the SUPERPLOT News System, which is accessed by executing the program *NEWS* in workspace 3 *SUPERPLOT*.

Whither SUPERPLOT?

SUPERPLOT will continue to evolve in increasing sophistication and flexibility, provided the feedback from the user community continues to be high. As mentioned in the Introduction to this paper, SUPERPLOT is primarily a tool for automating an activity previously performed by hand, and an increasing awareness by the SUPERPLOT development staff of contemporary graphics techniques and styles is essential to the long term success and popularity of SUPERPLOT.

Reference

[1] Keith, D.A., **Data Manipulation with MAGIC,** Proceedings of the 1980 APL Users Meeting, October, 1980.

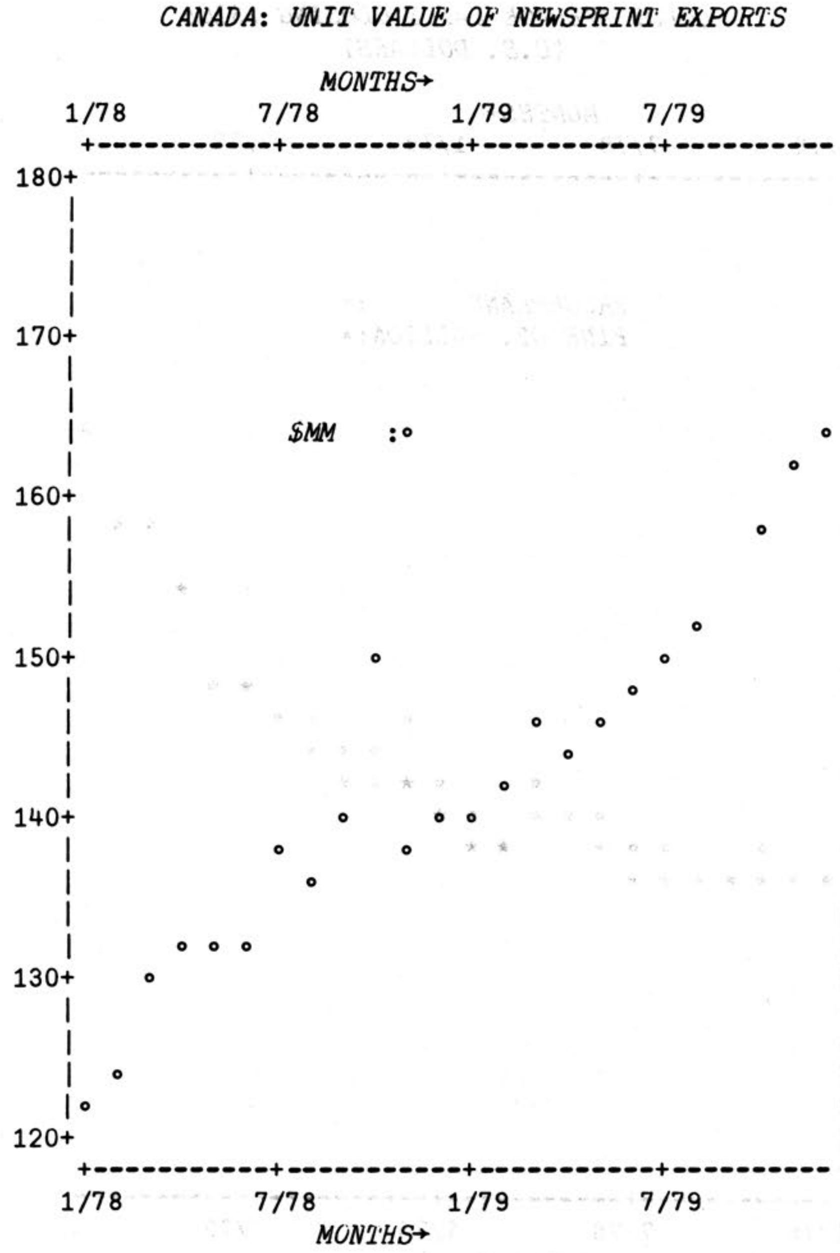

CANADA: UNIT VALUE OF NEWSPRINT EXPORTS
MONTHS→
1/78
7/78
1/79
7/79
180+
170+
160+
150+
140+
130+
120+
$MM :o
1/78
7/78
1/79
7/79
MONTHS→

Figure 1

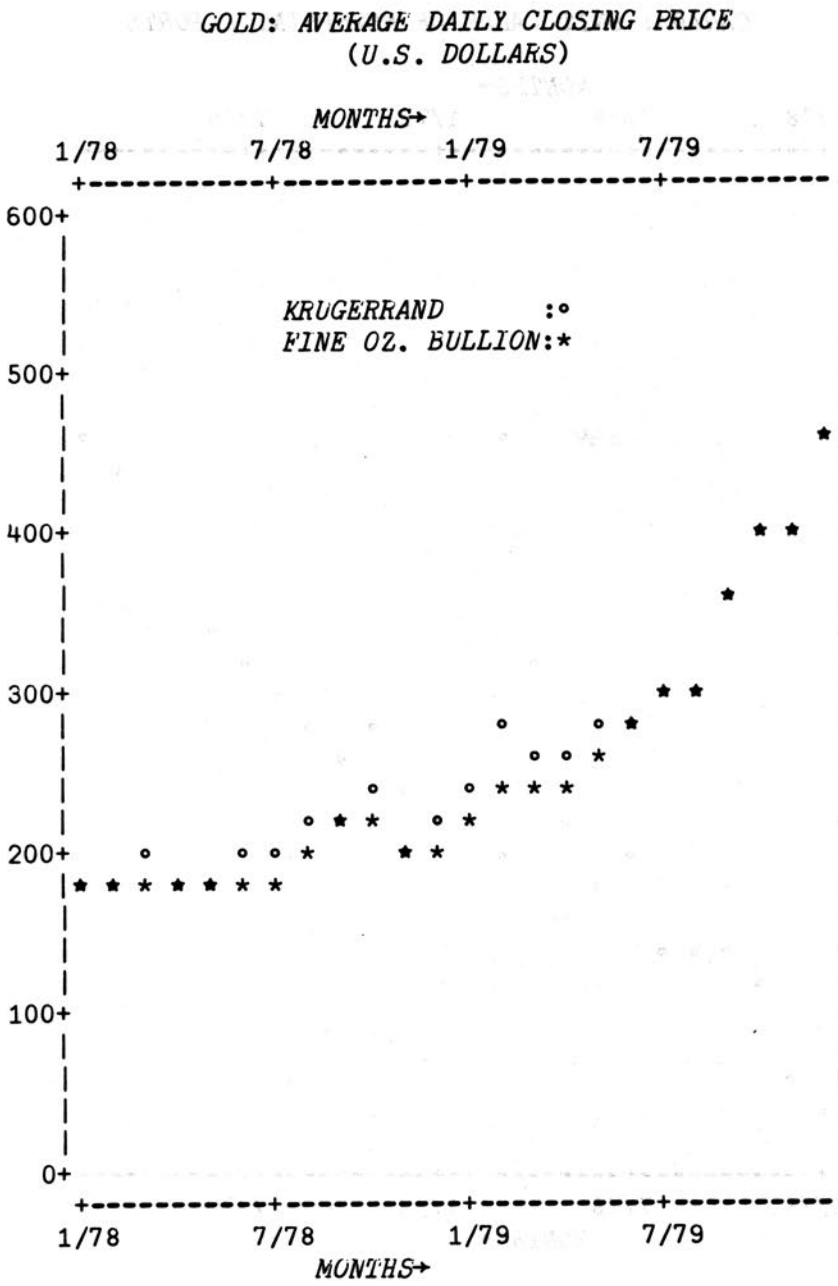

GOLD: AVERAGE DAILY CLOSING PRICE
(U.S. DOLLARS)
MONTHS→
1/78
7/78
1/79
7/79
600+
500+
400+
300+
200+
100+
0+
KRUGERRAND :○
FINE OZ. BULLION:*
1/78
7/78
1/79
7/79
MONTHS→

Figure 2

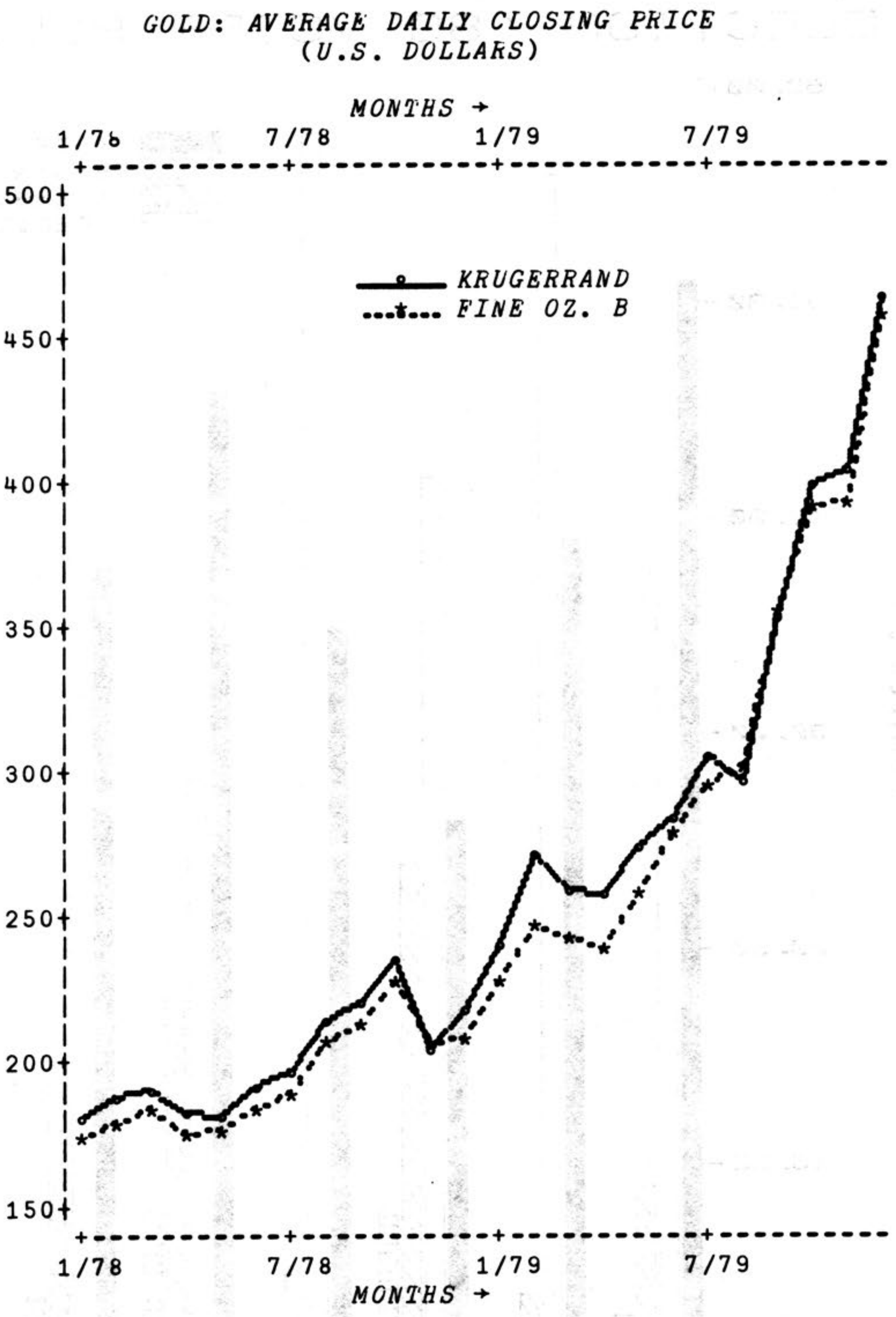

GOLD: AVERAGE DAILY CLOSING PRICE
(U.S. DOLLARS)
MONTHS →
1/78
7/78
1/79
7/79
500
450
400
350
300
250
200
150
KRUGERRAND
FINE OZ. B
1/78
7/78
1/79
7/79
MONTHS →

Figure 3

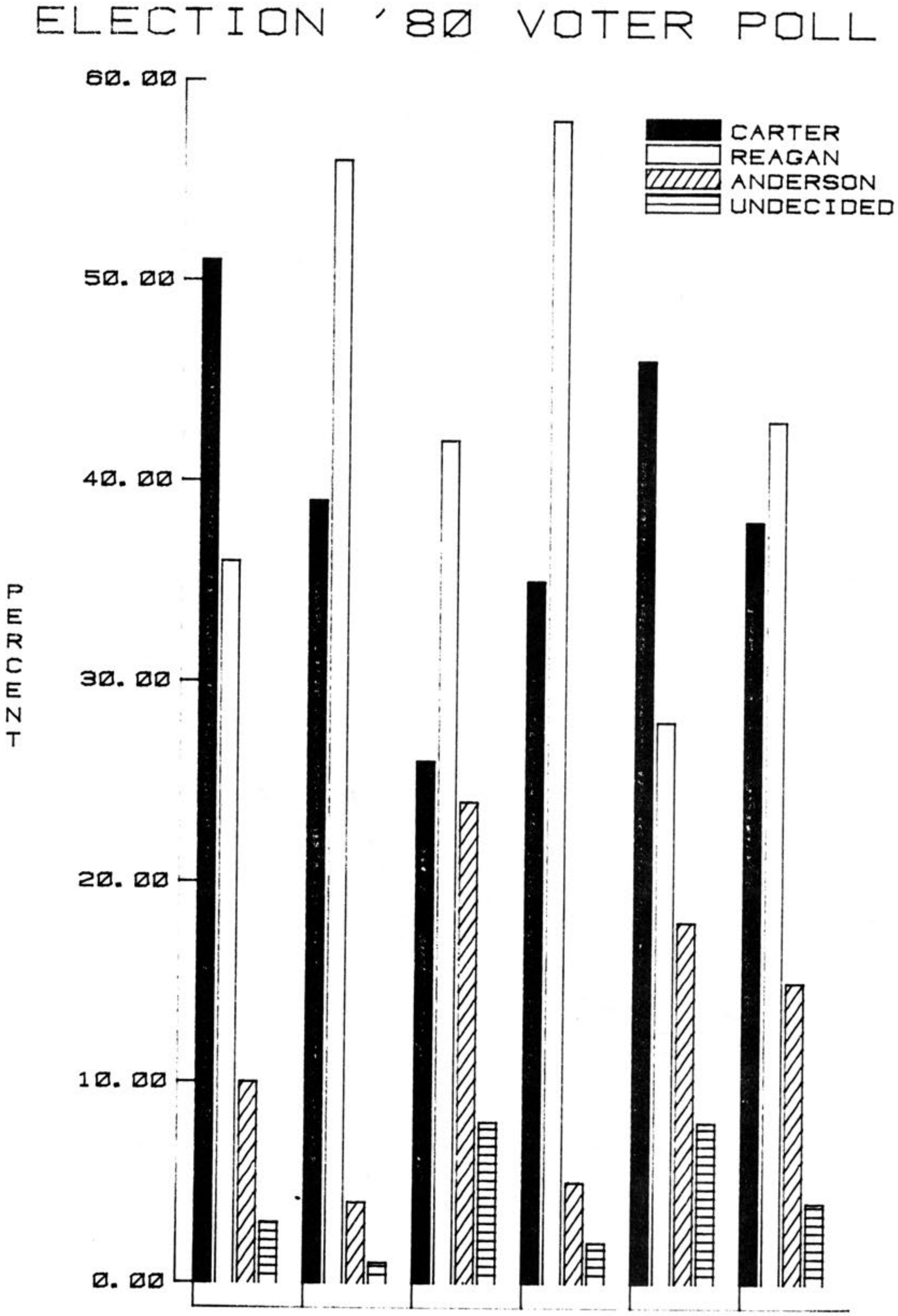
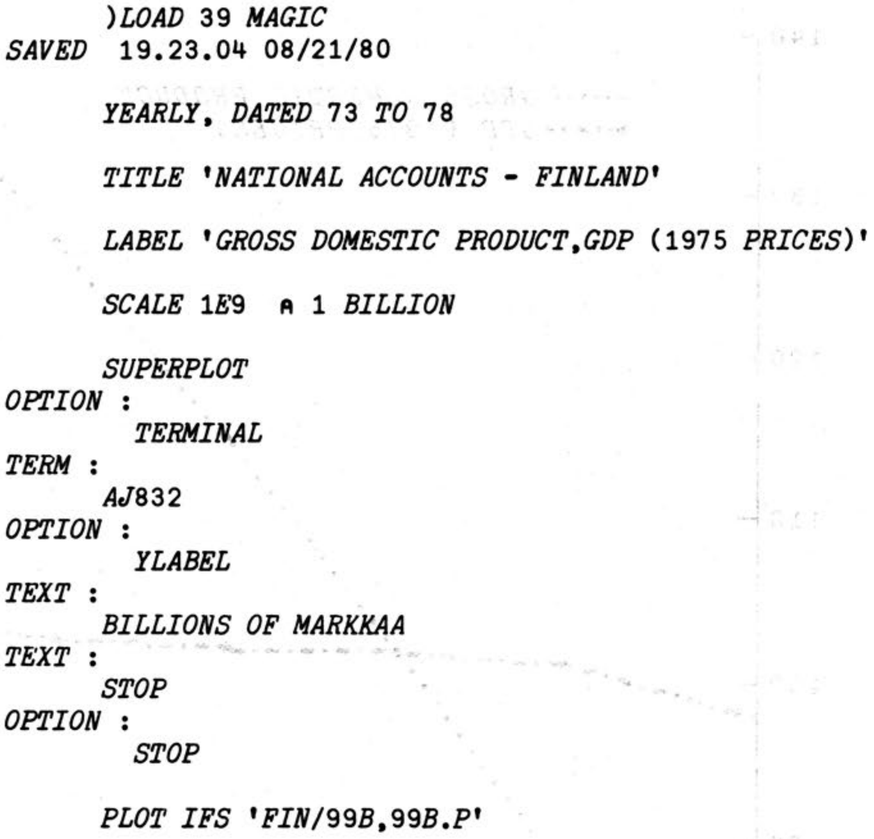

ELECTION '80 VOTER POLL
60.00
50.00
40.00
30.00
20.00
10.00
0.00
PERCENT
CARTER
REAGAN
ANDERSON
UNDECIDED
N. E.
S. E.
N. W.
S. W.
OTHER
TOTAL

Figure 4



```
      )LOAD 39 MAGIC
SAVED  19.23.04 08/21/80

      YEARLY, DATED 73 TO 78

      TITLE 'NATIONAL ACCOUNTS - FINLAND'

      LABEL 'GROSS DOMESTIC PRODUCT,GDP (1975 PRICES)'

      SCALE 1E9  ⍝ 1 BILLION

      SUPERPLOT
OPTION :
        TERMINAL
TERM :
      AJ832
OPTION :
        YLABEL
TEXT :
      BILLIONS OF MARKKAA
TEXT :
      STOP
OPTION :
        STOP

      PLOT IFS 'FIN/99B,99B.P'
```

Figure 5

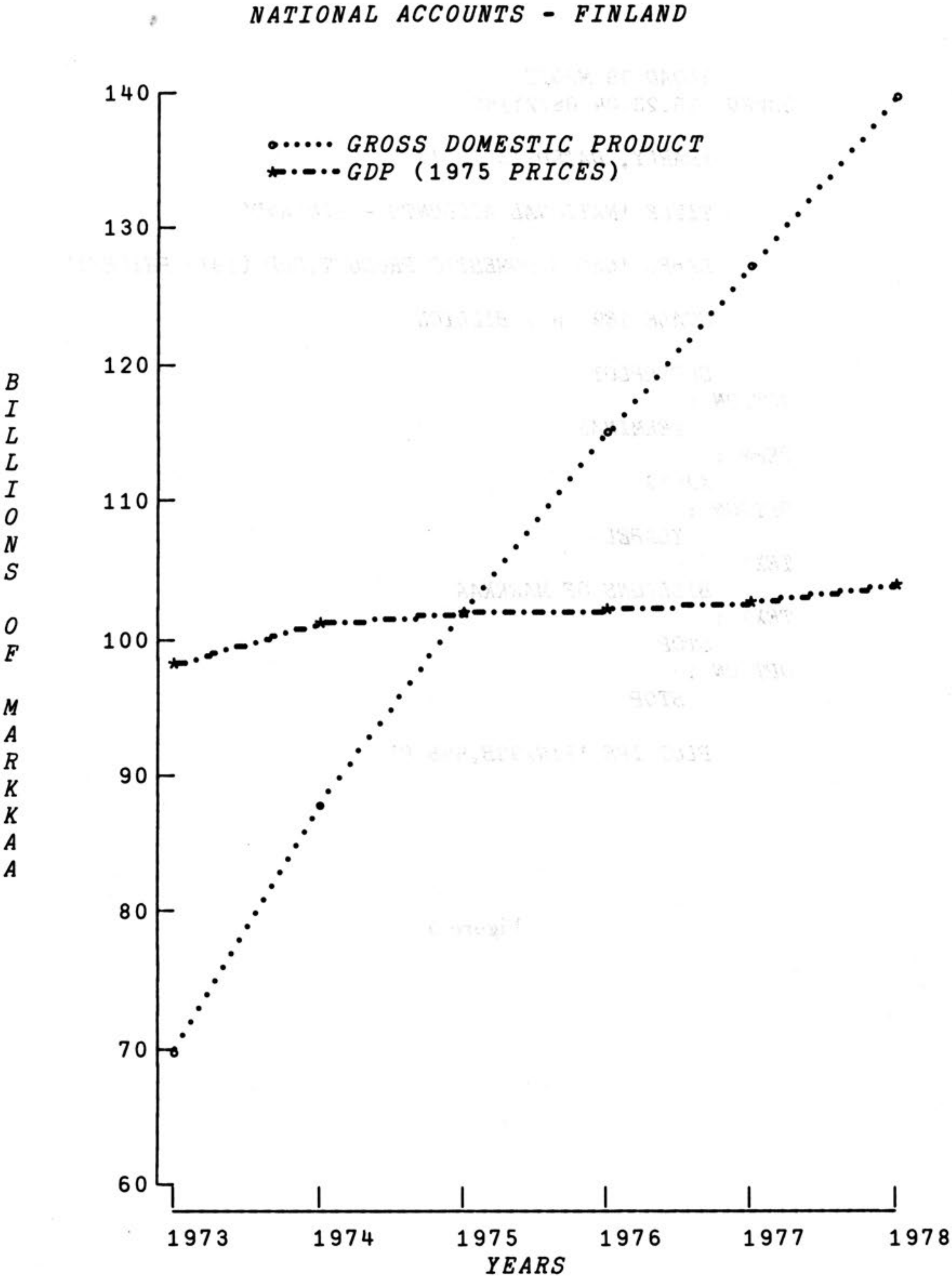
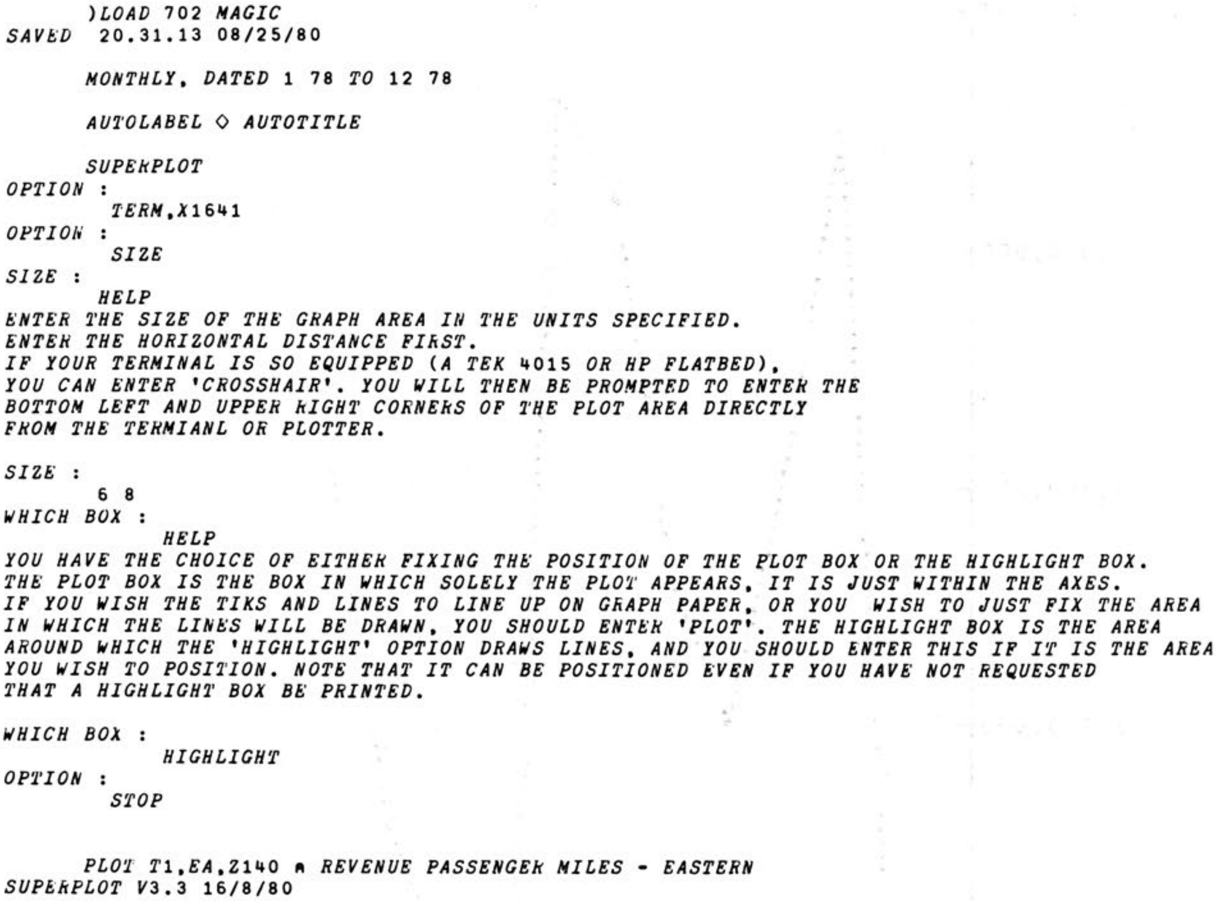


Figure 5 (con't)



```
      )LOAD 702 MAGIC
SAVED  20.31.13 08/25/80

      MONTHLY, DATED 1 78 TO 12 78

      AUTOLABEL ◇ AUTOTITLE

      SUPERPLOT
OPTION :
        TERM,X1641
OPTION :
        SIZE
SIZE :
        HELP
ENTER THE SIZE OF THE GRAPH AREA IN THE UNITS SPECIFIED.
ENTER THE HORIZONTAL DISTANCE FIRST.
IF YOUR TERMINAL IS SO EQUIPPED (A TEK 4015 OR HP FLATBED),
YOU CAN ENTER 'CROSSHAIR'. YOU WILL THEN BE PROMPTED TO ENTER THE
BOTTOM LEFT AND UPPER RIGHT CORNERS OF THE PLOT AREA DIRECTLY
FROM THE TERMIANL OR PLOTTER.

SIZE :
        6 8
WHICH BOX :
            HELP
YOU HAVE THE CHOICE OF EITHER FIXING THE POSITION OF THE PLOT BOX OR THE HIGHLIGHT BOX.
THE PLOT BOX IS THE BOX IN WHICH SOLELY THE PLOT APPEARS, IT IS JUST WITHIN THE AXES.
IF YOU WISH THE TIKS AND LINES TO LINE UP ON GRAPH PAPER, OR YOU  WISH TO JUST FIX THE AREA
IN WHICH THE LINES WILL BE DRAWN, YOU SHOULD ENTER 'PLOT'. THE HIGHLIGHT BOX IS THE AREA
AROUND WHICH THE 'HIGHLIGHT' OPTION DRAWS LINES, AND YOU SHOULD ENTER THIS IF IT IS THE AREA
YOU WISH TO POSITION. NOTE THAT IT CAN BE POSITIONED EVEN IF YOU HAVE NOT REQUESTED
THAT A HIGHLIGHT BOX BE PRINTED.

WHICH BOX :
            HIGHLIGHT
OPTION :
        STOP


      PLOT T1,EA,Z140 ⍝ REVENUE PASSENGER MILES - EASTERN
SUPERPLOT V3.3 16/8/80
```

Figure 6

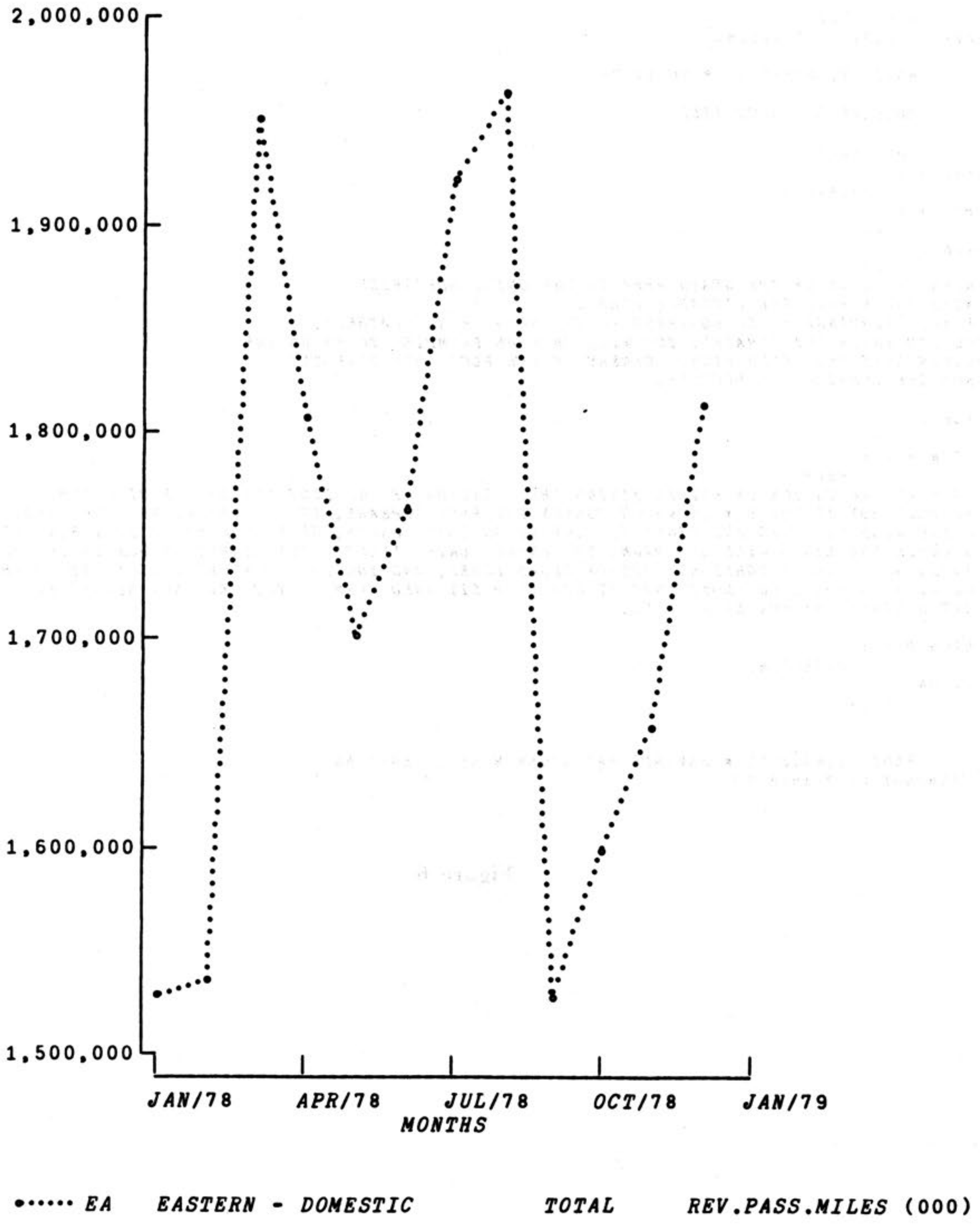
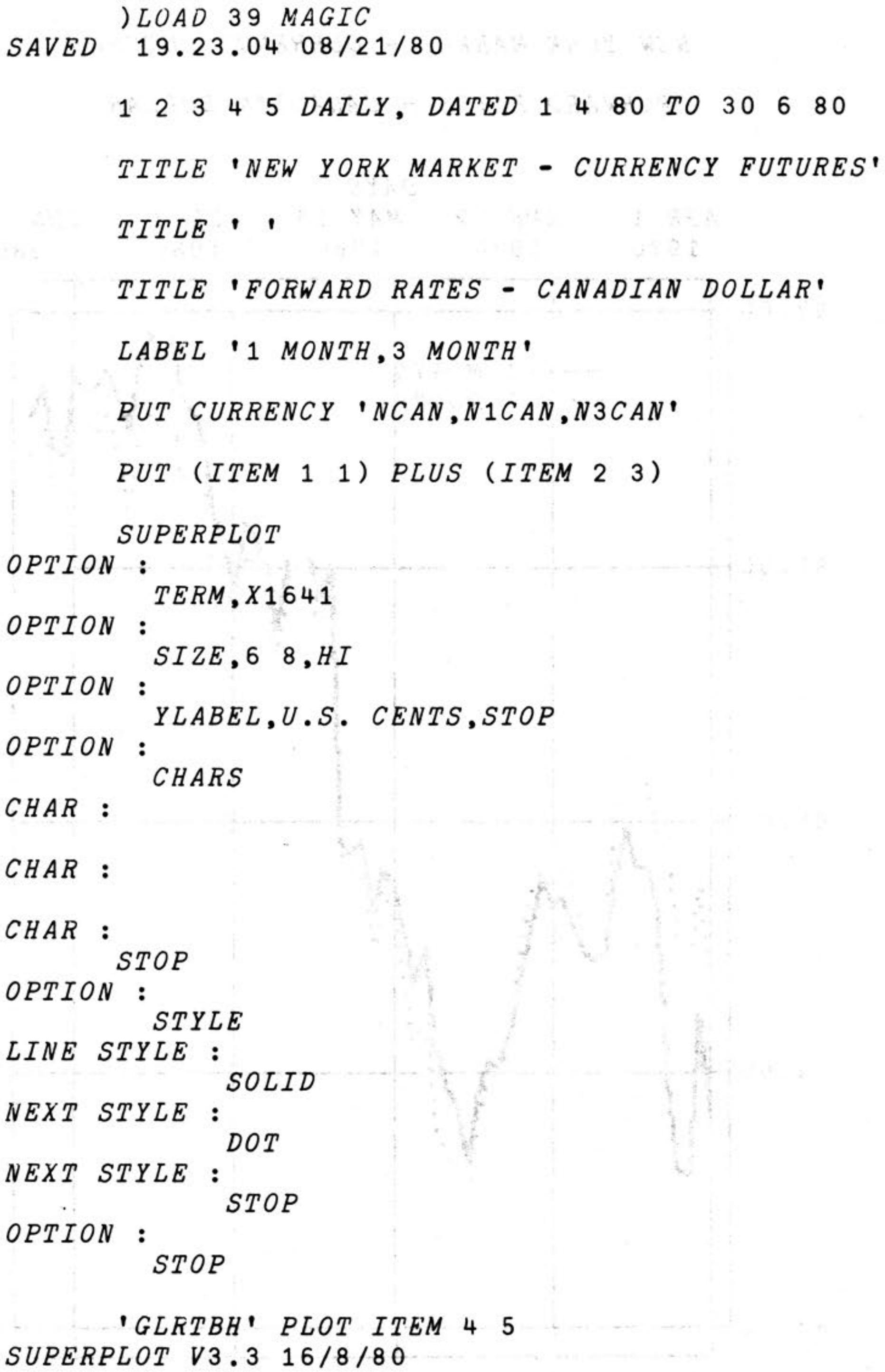


Figure 6 (con't)



```
      )LOAD 39 MAGIC
SAVED  19.23.04 08/21/80

      1 2 3 4 5 DAILY, DATED 1 4 80 TO 30 6 80

      TITLE 'NEW YORK MARKET - CURRENCY FUTURES'

      TITLE ' '

      TITLE 'FORWARD RATES - CANADIAN DOLLAR'

      LABEL '1 MONTH,3 MONTH'

      PUT CURRENCY 'NCAN,N1CAN,N3CAN'

      PUT (ITEM 1 1) PLUS (ITEM 2 3)

      SUPERPLOT
OPTION :
        TERM,X1641
OPTION :
        SIZE,6 8,HI
OPTION :
        YLABEL,U.S. CENTS,STOP
OPTION :
        CHARS
CHAR :

CHAR :

CHAR :
      STOP
OPTION :
        STYLE
LINE STYLE :
            SOLID
NEXT STYLE :
            DOT
NEXT STYLE :
            STOP
OPTION :
        STOP

      'GLRTBH' PLOT ITEM 4 5
SUPERPLOT V3.3 16/8/80
```

Figure 7

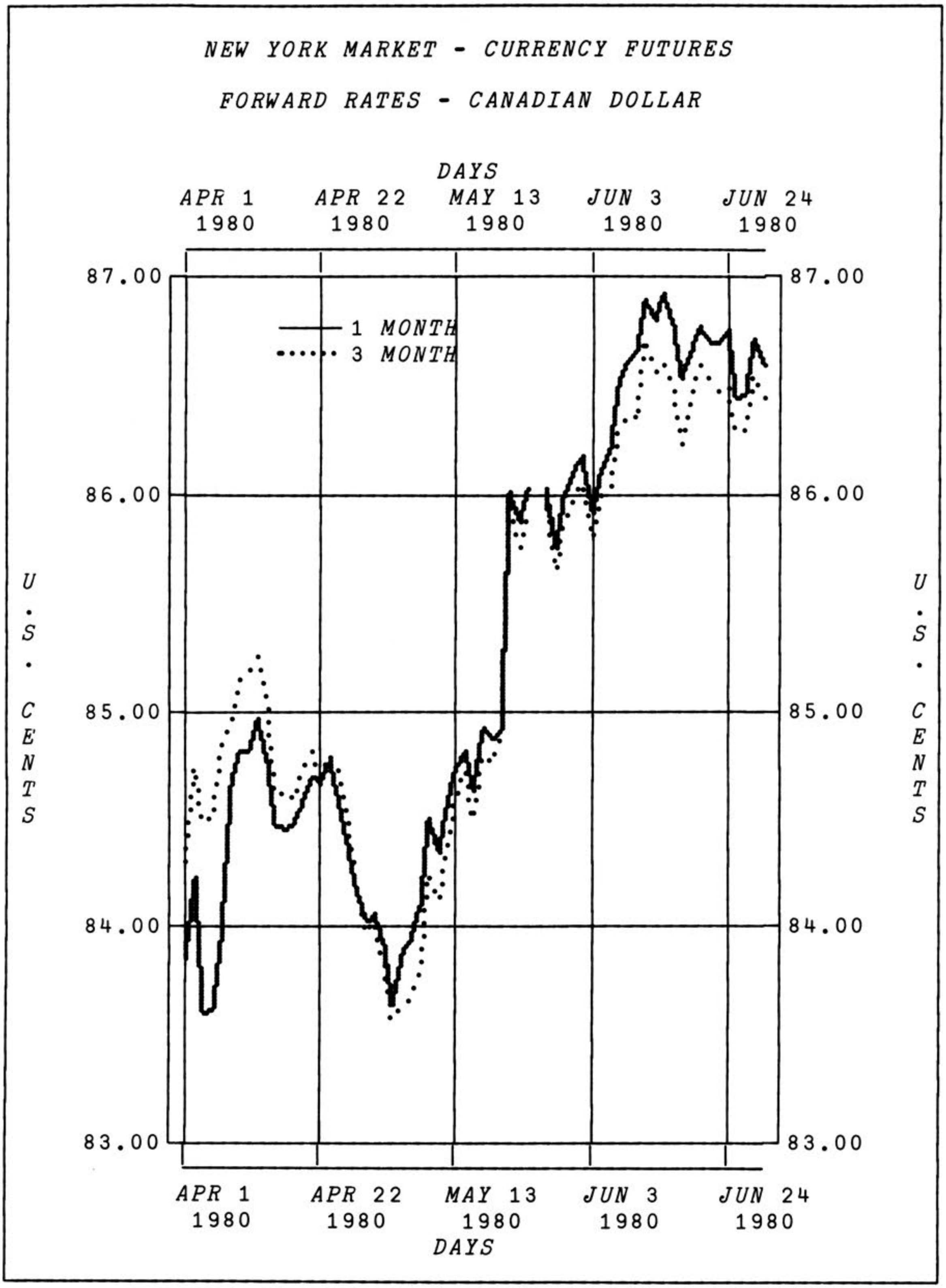
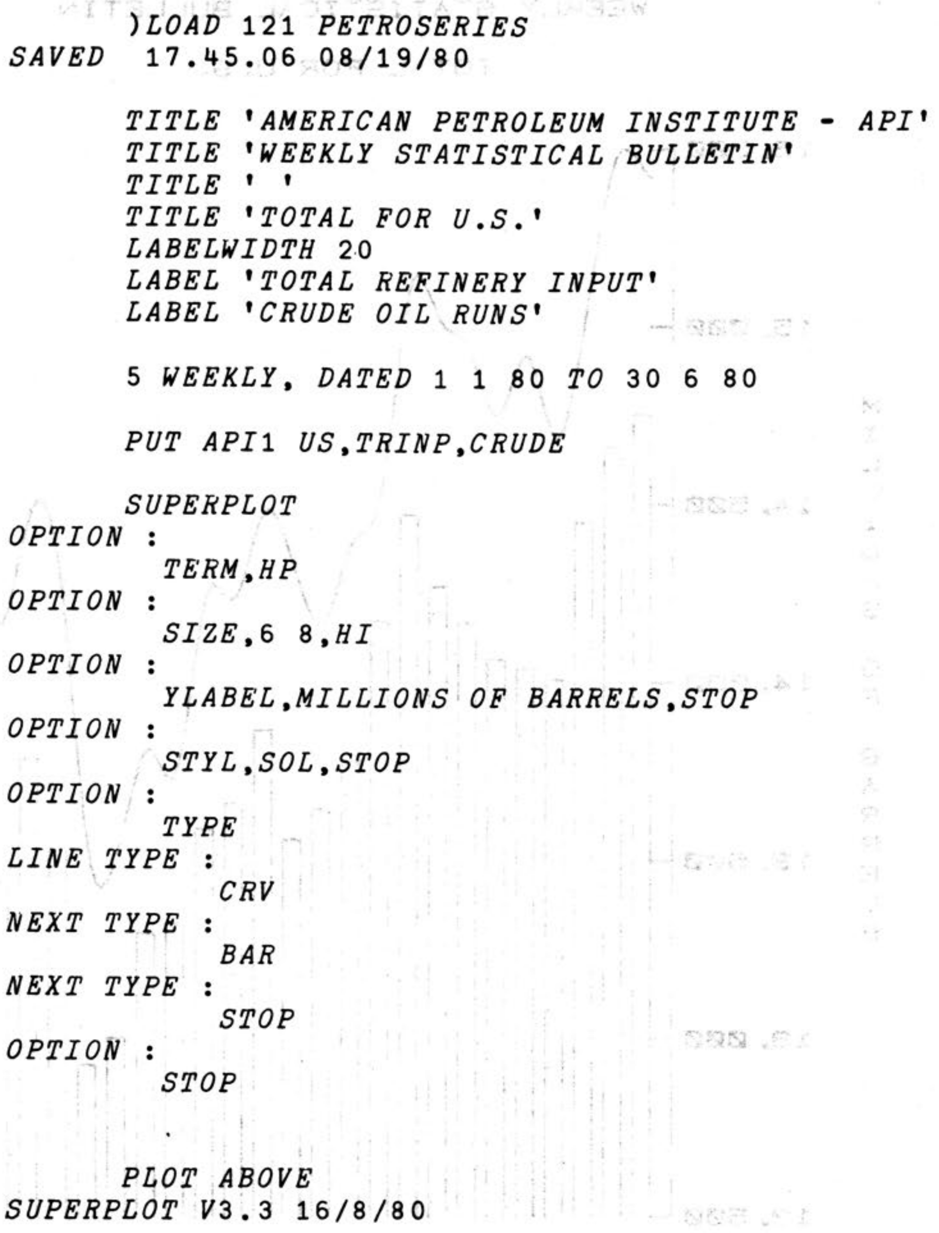


Figure 7 (con't)



```
        )LOAD 121 PETROSERIES
SAVED  17.45.06 08/19/80

        TITLE 'AMERICAN PETROLEUM INSTITUTE - API'
        TITLE 'WEEKLY STATISTICAL BULLETIN'
        TITLE ' '
        TITLE 'TOTAL FOR U.S.'
        LABELWIDTH 20
        LABEL 'TOTAL REFINERY INPUT'
        LABEL 'CRUDE OIL RUNS'

        5 WEEKLY, DATED 1 1 80 TO 30 6 80

        PUT API1 US,TRINP,CRUDE

        SUPERPLOT
OPTION :
          TERM,HP
OPTION :
          SIZE,6 8,HI
OPTION :
          YLABEL,MILLIONS OF BARRELS,STOP
OPTION :
          STYL,SOL,STOP
OPTION :
          TYPE
LINE TYPE :
             CRV
NEXT TYPE :
             BAR
NEXT TYPE :
             STOP
OPTION :
          STOP

              .

        PLOT ABOVE
SUPERPLOT V3.3 16/8/80
```

Figure 8

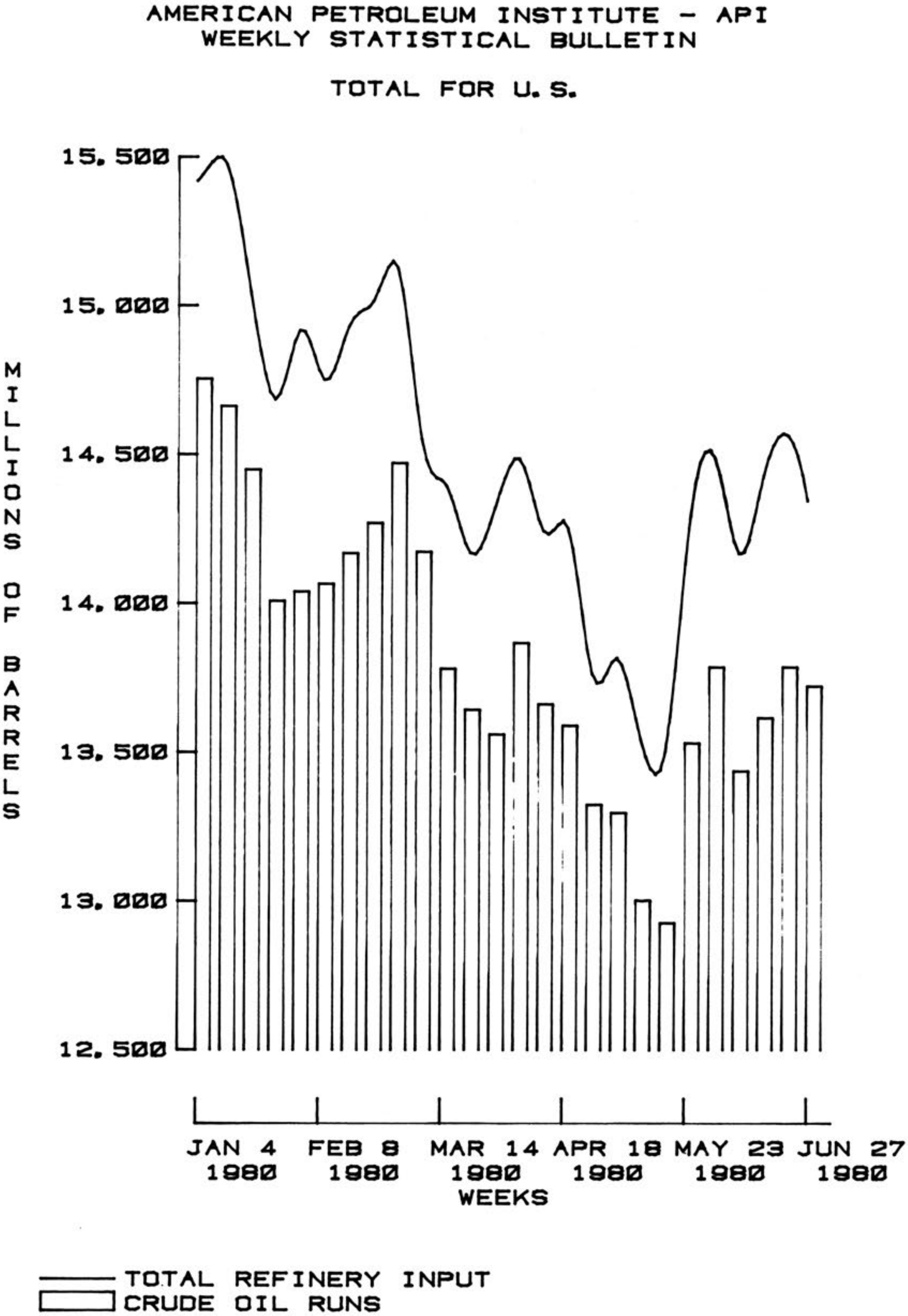


Figure 8 (con't)

```
      )LOAD 39 MAGIC
SAVED  19.23.04 08/21/80

      1 2 3 4 5 DAILY, DATED 15 8 79 TO 31 8 80

      TITLE 'OCELOT.B - A SPECULATOR''S DREAM'
      TITLE 'EXCHANGE SYMBOL: OIL.B'
      LABEL 'HI/LO/CLOSE,,,VOLUME'
      SUPERPLOT
OPTION :
        TERM,HP,SIZE,6 8,HI
OPTION :
        GROUP
GROUP NUMBER :
               1
LINE NOS. :
           1 2 3
∘/∘ OF GRAPH :
             65
GROUP NUMBER :
               2
LINE NOS. :
           4
∘/∘ OF GRAPH :
             35
GROUP NUMBER :
               STOP
OPTION :
        LINE
NUMBER(S) :
           1 2 3
TYPE :
      HLC
COLOUR :
        BLACK
STYLE  :
        SOLID
CHAR :
      STOP
NUMBER(S) :
           4
TYPE :
      HI3
COLOUR :
        BLA
STYLE  :
        SOL
CHAR :
      STOP
NUMBER(S) :
           STOP
OPTION :
        STOP

      'LRTBH' PLOT 'OIL.B' FPDAILY 2 3 4 1 A TSE ISSUE
SUPERPLOT V3.3 16/8/80
```

Figure 9

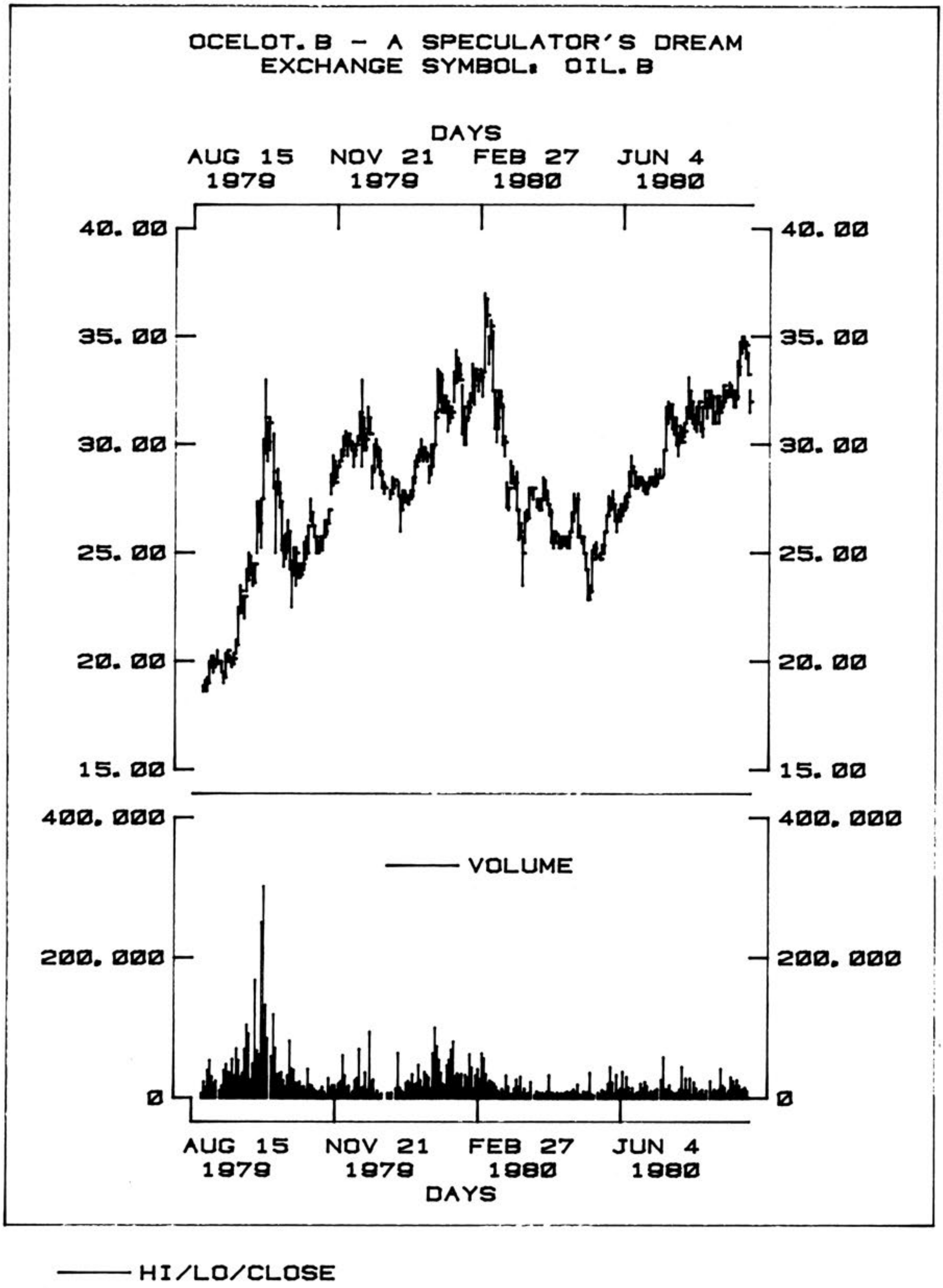


Figure 9 (con't)

```
      )LOAD 702 INS
SAVED    1.04.03 08/25/80

      TITLE 'NEW YORK - LONDON'
      TITLE 'PASSENGERS BY FLAG OF CARRIER'
      LABEL 'INDIA,U.K.,U.S.'
      PUT 'NYC-LON' INS IND,UK,USA,TOTPAX

      SUPERPLOT
OPTION :
        TERM,CALC81,SIZE,6 9,HI
OPTION :
        YLABEL,PASSENGERS-ALL CITIZENSHIPS,STOP
OPTION :
        LINE,1 2 3,SHI,BLACK,SOLID,STOP
NUMBER(S) :
             STOP
OPTION :
        FONT
WHICH TEXT :
              LEGEND
CHARACTER SIZE :
                  2
WHICH TEXT :
              STOP
OPTION :
        SHADING
LINE NO. :
          1
TO :
    BOTTOM
ANGLE :
       45
SPACING :
         10
COLOUR :
        BLACK
STYLE  :
        SOLID
LINE NO. :
          2,1,90,6,BLA,SOL
LINE NO. :
          3,2,135,10,BLA,SOL,STOP,STOP

      PLOT ABOVE
SUPERPLOT V3.3 16/8/80
```

Figure 10

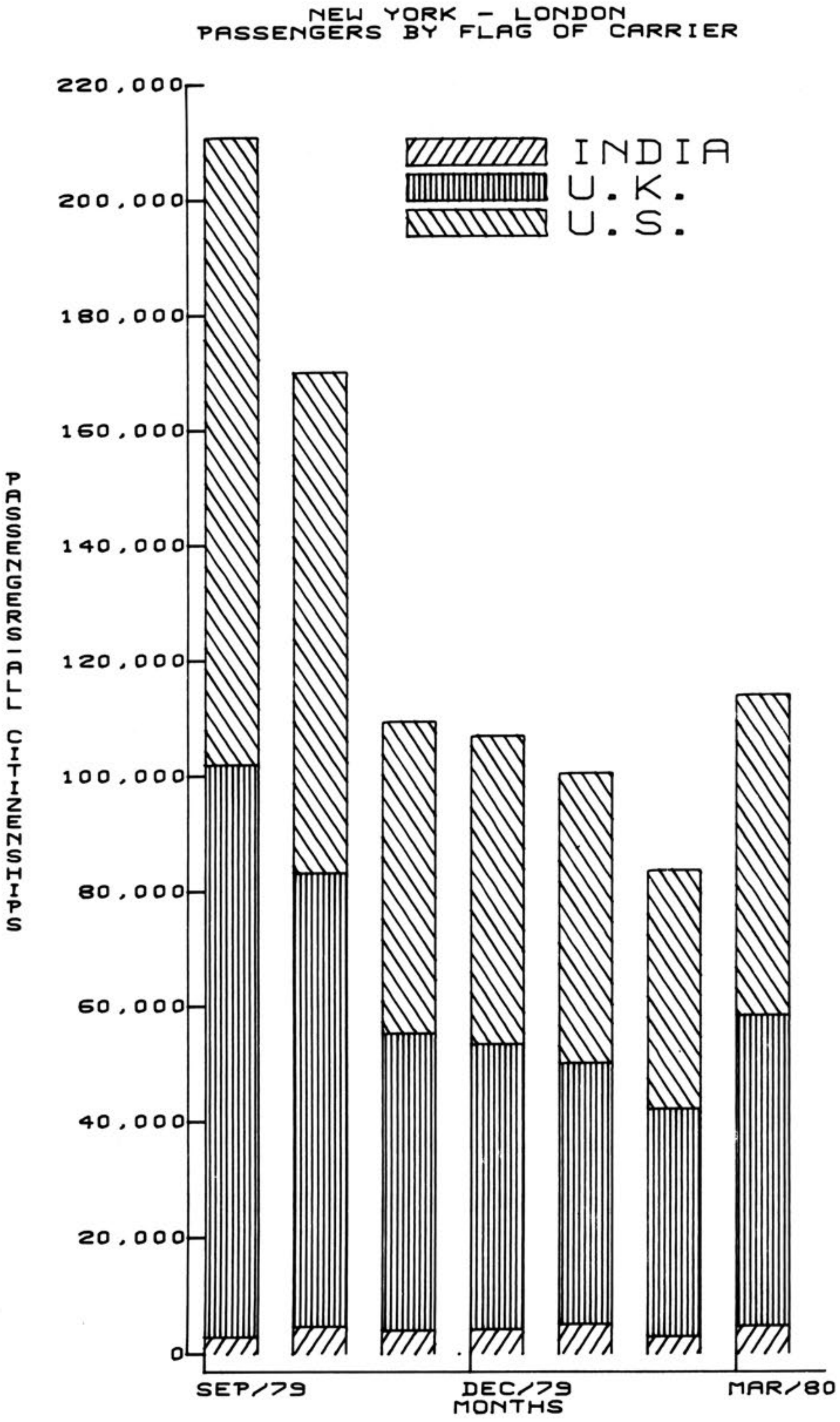
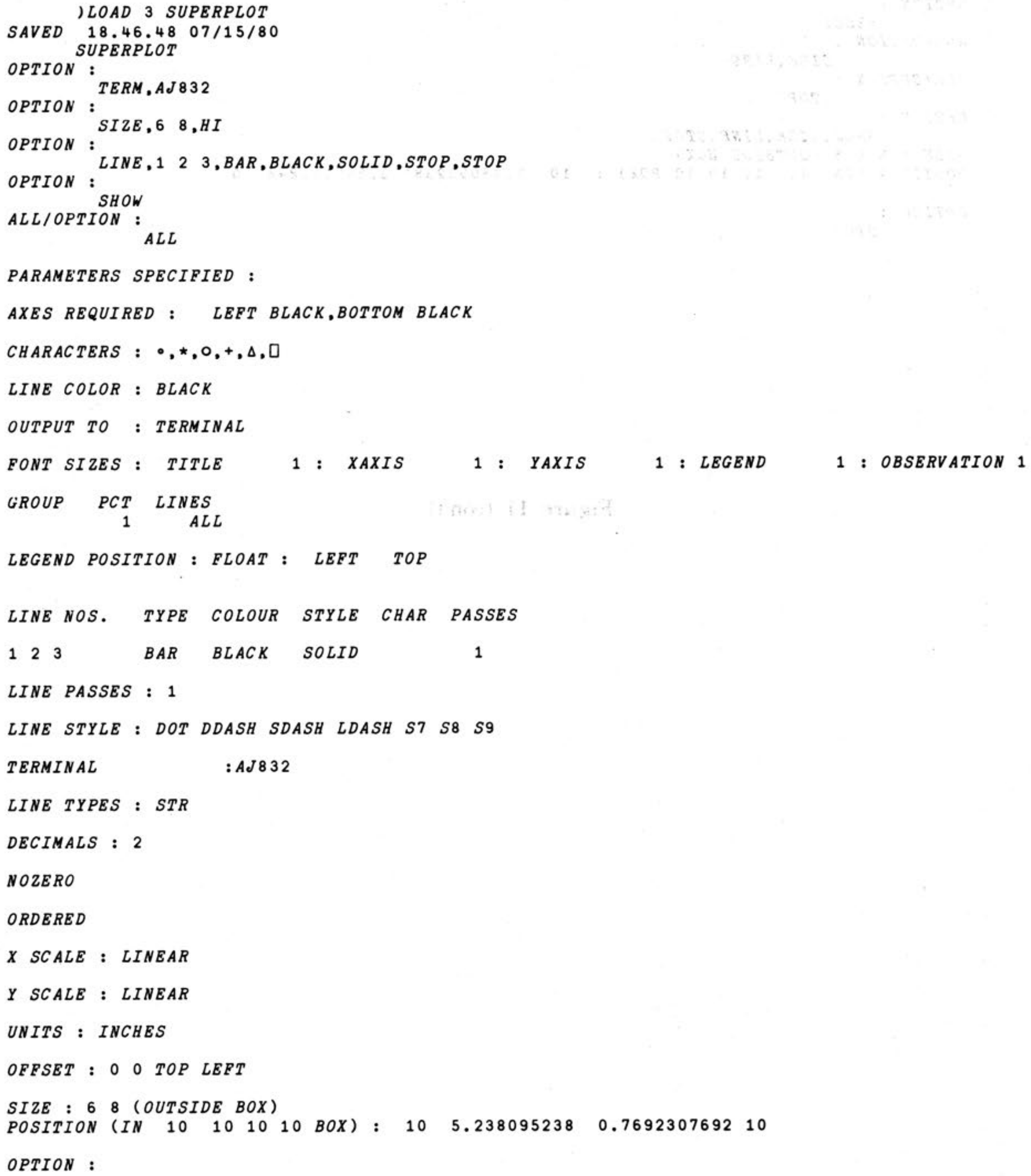
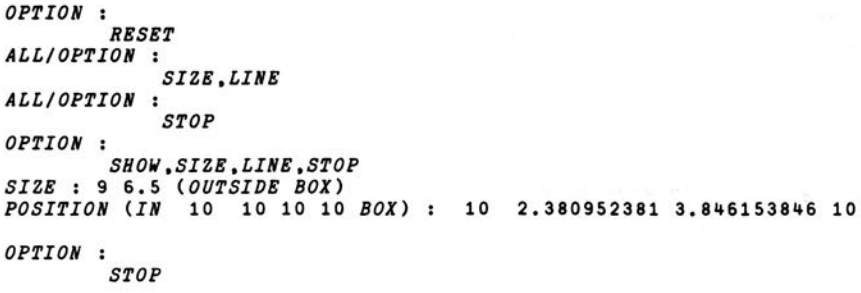


Figure 10 (con't)



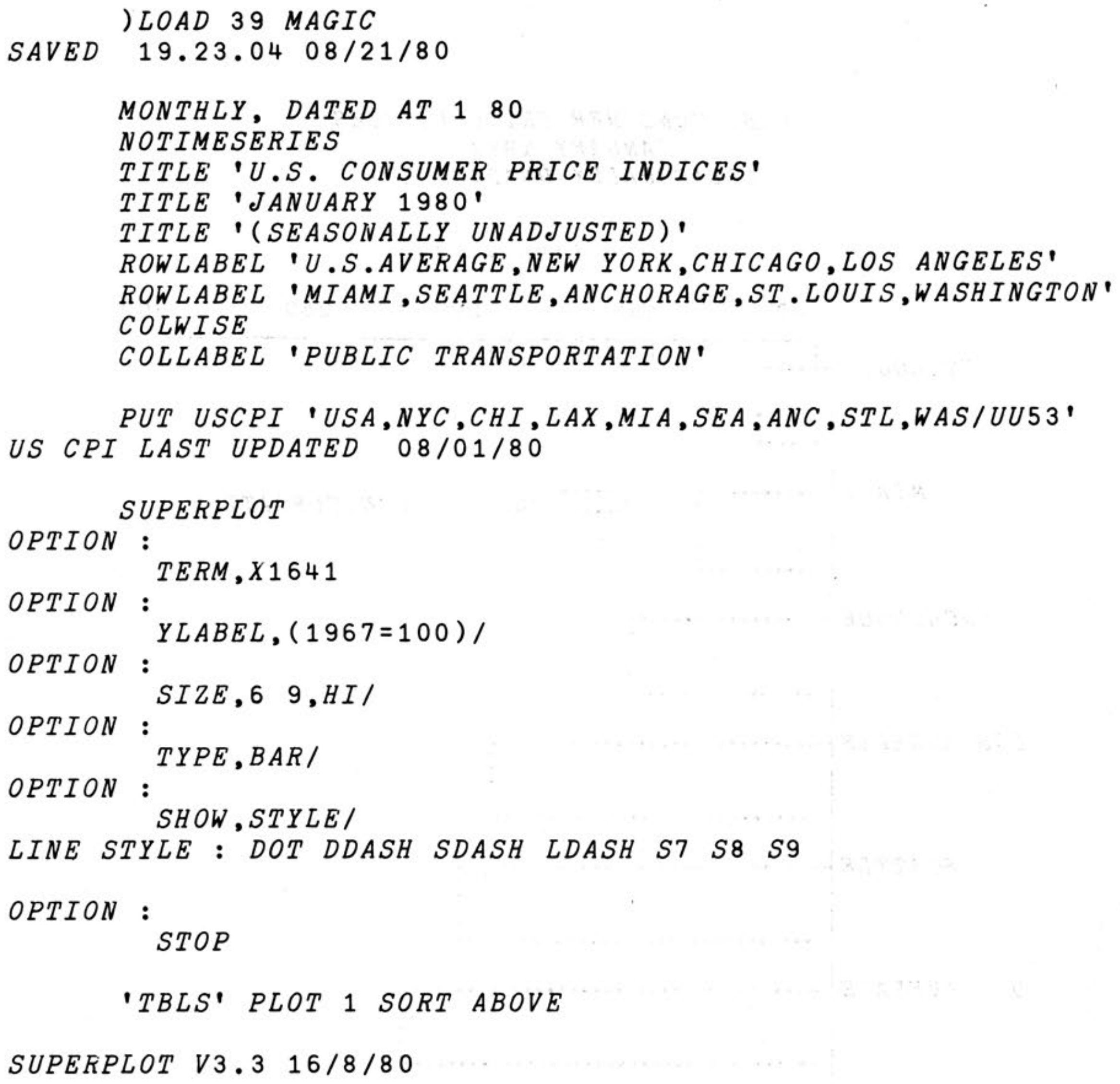
```
      )LOAD 3 SUPERPLOT
SAVED  18.46.48 07/15/80
      SUPERPLOT
OPTION :
       TERM,AJ832
OPTION :
       SIZE,6 8,HI
OPTION :
       LINE,1 2 3,BAR,BLACK,SOLID,STOP,STOP
OPTION :
       SHOW
ALL/OPTION :
          ALL

PARAMETERS SPECIFIED :

AXES REQUIRED :   LEFT BLACK,BOTTOM BLACK

CHARACTERS : •,*,○,+,Δ,□

LINE COLOR : BLACK

OUTPUT TO  : TERMINAL

FONT SIZES :  TITLE      1 :  XAXIS      1 :  YAXIS      1 : LEGEND      1 : OBSERVATION 1

GROUP   PCT  LINES
          1     ALL

LEGEND POSITION : FLOAT :  LEFT   TOP


LINE NOS.   TYPE  COLOUR  STYLE  CHAR  PASSES

1 2 3       BAR   BLACK   SOLID          1

LINE PASSES : 1

LINE STYLE : DOT DDASH SDASH LDASH S7 S8 S9

TERMINAL            :AJ832

LINE TYPES : STR

DECIMALS : 2

NOZERO

ORDERED

X SCALE : LINEAR

Y SCALE : LINEAR

UNITS : INCHES

OFFSET : 0 0 TOP LEFT

SIZE : 6 8 (OUTSIDE BOX)
POSITION (IN  10  10 10 10 BOX) :  10  5.238095238  0.7692307692 10

OPTION :
```

Figure 11



```
OPTION :
        RESET
ALL/OPTION :
            SIZE,LINE
ALL/OPTION :
            STOP
OPTION :
        SHOW,SIZE,LINE,STOP
SIZE : 9 6.5 (OUTSIDE BOX)
POSITION (IN  10  10 10 10 BOX) :  10  2.380952381 3.846153846 10

OPTION :
        STOP
```

Figure 11 (con't)



```
      )LOAD 39 MAGIC
SAVED  19.23.04 08/21/80

      MONTHLY, DATED AT 1 80
      NOTIMESERIES
      TITLE 'U.S. CONSUMER PRICE INDICES'
      TITLE 'JANUARY 1980'
      TITLE '(SEASONALLY UNADJUSTED)'
      ROWLABEL 'U.S.AVERAGE,NEW YORK,CHICAGO,LOS ANGELES'
      ROWLABEL 'MIAMI,SEATTLE,ANCHORAGE,ST.LOUIS,WASHINGTON'
      COLWISE
      COLLABEL 'PUBLIC TRANSPORTATION'

      PUT USCPI 'USA,NYC,CHI,LAX,MIA,SEA,ANC,STL,WAS/UU53'
US CPI LAST UPDATED  08/01/80

      SUPERPLOT
OPTION :
        TERM,X1641
OPTION :
        YLABEL,(1967=100)/
OPTION :
        SIZE,6 9,HI/
OPTION :
        TYPE,BAR/
OPTION :
        SHOW,STYLE/
LINE STYLE : DOT DDASH SDASH LDASH S7 S8 S9

OPTION :
        STOP

      'TBLS' PLOT 1 SORT ABOVE

SUPERPLOT V3.3 16/8/80
```

Figure 12

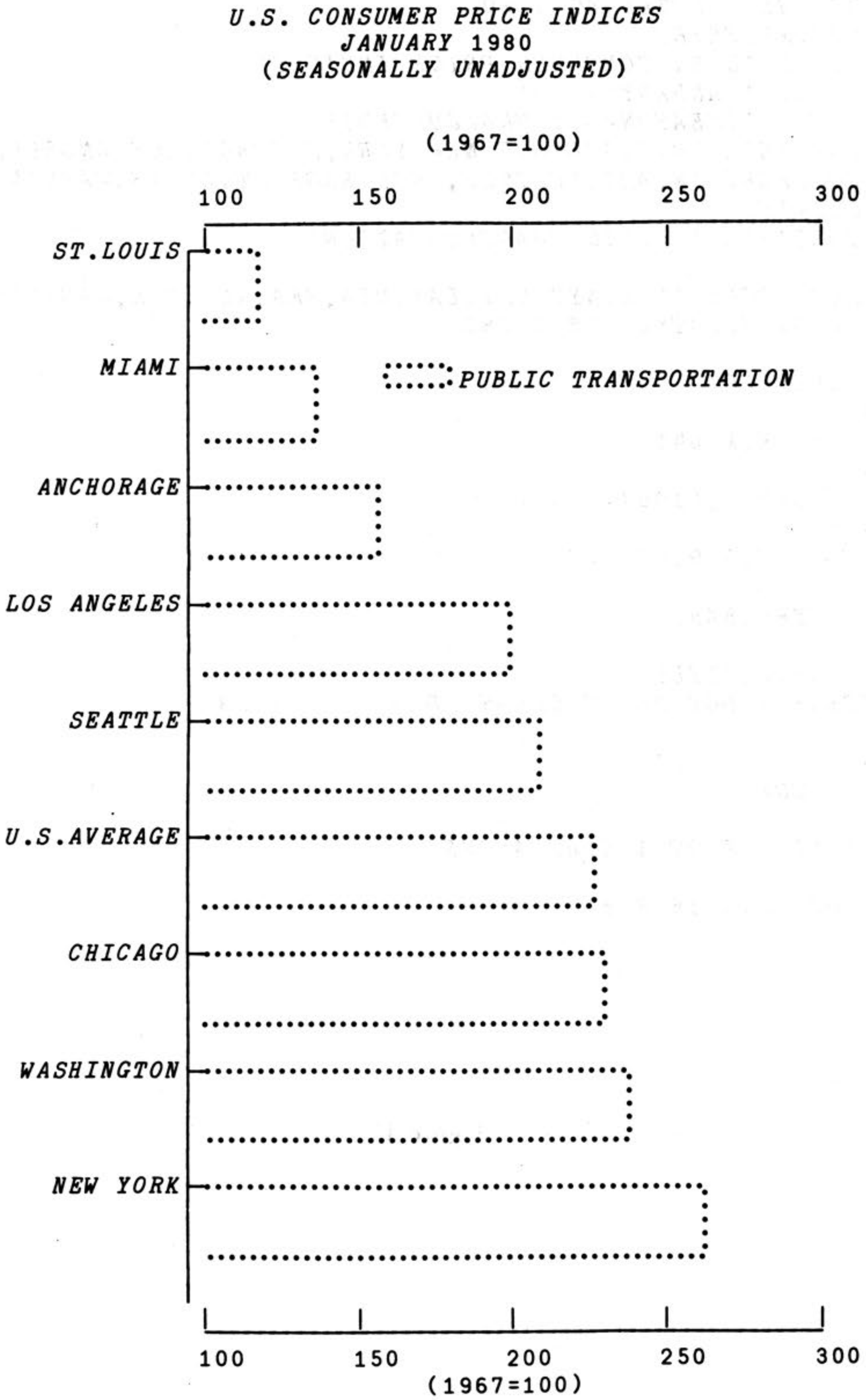


Figure 12 (con't)

```
      )LOAD 39 MAGIC
SAVED  19.23.04 08/21/80

      YEARLY, DATED 66 TO 79
      LASTVALUE
      POPCAN←CANSIM 'D1' ⍝ POPULATION OF CANADA
CANSIM MINIBASE LAST UPDATED WITH DATA RELEASED 80/08/29  13:50:35

      BANKSREVS←1 2 5 YBANK 4 ⍝ BANKS' TOTAL REVENUES

      NOTIMESERIES
      ROWWISE
      TITLE 'CORRELATION BETWEEN BANK REVENUES AND POPULATION'
      SUPERPLOT
OPTION :
        TERM,X1641
OPTION :
        SIZE,6 9,HI/
OPTION :
        YLABEL,TOTAL REVENUE $000/
OPTION :
        XLABEL,POPULATION OF CANADA/
OPTION :
        LABEL
LINE :
       1
LABEL :
        BANK OF MONTREAL
LINE :
       2,BANK OF NOVA SCOTIA
LINE :
       3,BANK OF COMMERCE/
OPTION :
        TYPE,CRV/
OPTION :
        CHAR,,,/
OPTION :
        YSCALE
LIN OR LOG :
             LOG
OPTION :
        STOP

      'GLRB' PLOT BANKSREVS VERSUS POPCAN
SUPERPLOT V3.3 16/8/80
```

Figure 13

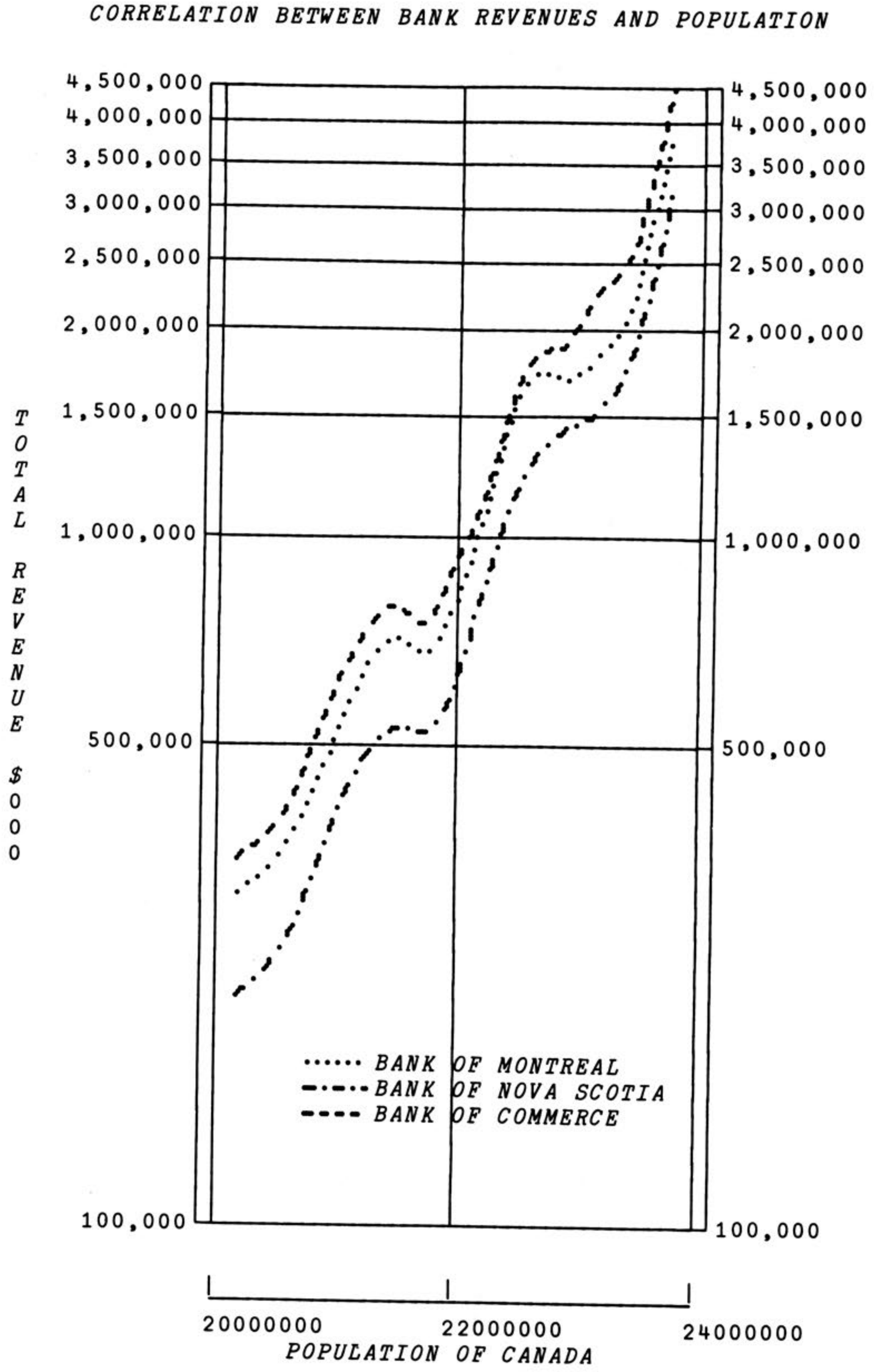
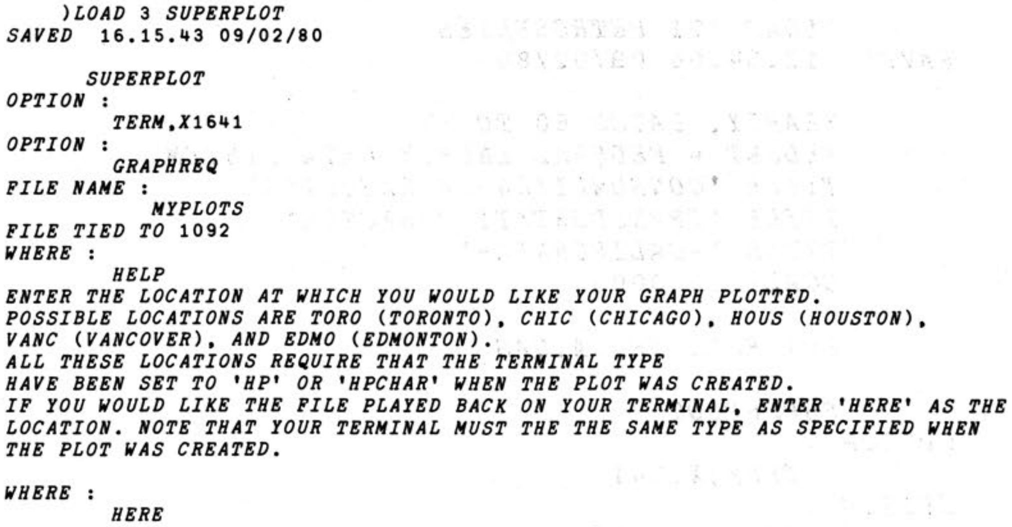


Figure 13 (con't)

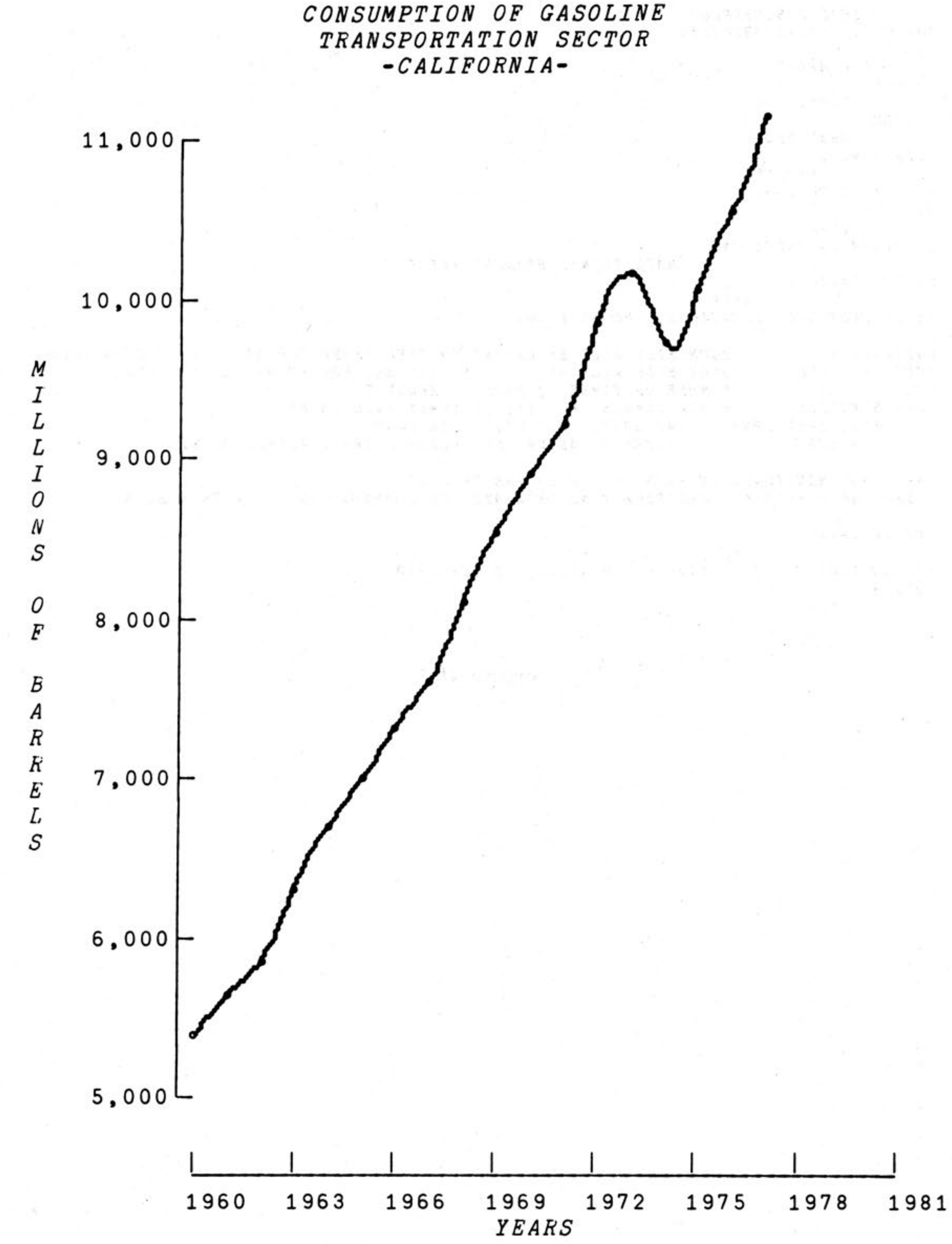
```
      )LOAD 121 PETROSERIES
SAVED  12.54.06 09/02/80

      YEARLY, DATED 60 TO 79
      FEDSET ⍝ FEDERAL ENERGY DATA SYSTEM
      TITLE 'CONSUMPTION OF GASOLINE'
      TITLE 'TRANSPORTATION SECTOR'
      TITLE '-CALIFORNIA-'
      SCALE 1000000

      PUT FEDS TR,CA,GAS

      SUPERPLOT
OPTION :
        TERM,X1641
OPTION :
        SIZE,6 8,HI
OPTION :
        YLABEL,MILLIONS OF BARRELS/
OPTION :
        STYLE,SOLID/
OPTION :
        TYPE,CRV/
OPTION :
        FILE
FILE NAME :
           MYPLOTS
OPTION :
        STOP

      PLOT ABOVE
SUPERPLOT V3.3 16/8/80
PLOT APPENDED TO FILE    1724097 MYPLOTS     AT 16:30
```

Figure 14



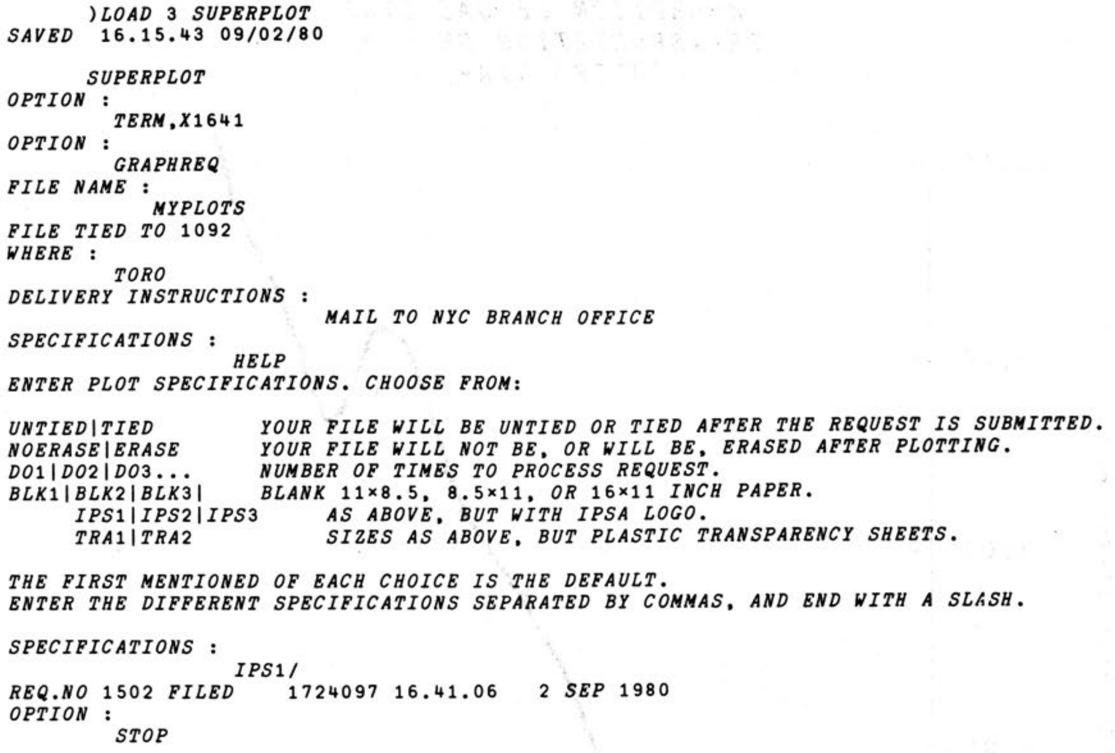
```
    )LOAD 3 SUPERPLOT
SAVED  16.15.43 09/02/80

      SUPERPLOT
OPTION :
        TERM,X1641
OPTION :
        GRAPHREQ
FILE NAME :
           MYPLOTS
FILE TIED TO 1092
WHERE :
        HELP
ENTER THE LOCATION AT WHICH YOU WOULD LIKE YOUR GRAPH PLOTTED.
POSSIBLE LOCATIONS ARE TORO (TORONTO), CHIC (CHICAGO), HOUS (HOUSTON),
VANC (VANCOVER), AND EDMO (EDMONTON).
ALL THESE LOCATIONS REQUIRE THAT THE TERMINAL TYPE
HAVE BEEN SET TO 'HP' OR 'HPCHAR' WHEN THE PLOT WAS CREATED.
IF YOU WOULD LIKE THE FILE PLAYED BACK ON YOUR TERMINAL, ENTER 'HERE' AS THE
LOCATION. NOTE THAT YOUR TERMINAL MUST THE THE SAME TYPE AS SPECIFIED WHEN
THE PLOT WAS CREATED.

WHERE :
        HERE
```

Figure 15



```
OPTION :
        STOP
```

Figure 15 (con't)



```
      )LOAD 3 SUPERPLOT
SAVED  16.15.43 09/02/80

      SUPERPLOT
OPTION :
        TERM,X1641
OPTION :
        GRAPHREQ
FILE NAME :
           MYPLOTS
FILE TIED TO 1092
WHERE :
        TORO
DELIVERY INSTRUCTIONS :
                        MAIL TO NYC BRANCH OFFICE
SPECIFICATIONS :
                 HELP
ENTER PLOT SPECIFICATIONS. CHOOSE FROM:

UNTIED|TIED          YOUR FILE WILL BE UNTIED OR TIED AFTER THE REQUEST IS SUBMITTED.
NOERASE|ERASE        YOUR FILE WILL NOT BE, OR WILL BE, ERASED AFTER PLOTTING.
DO1|DO2|DO3...       NUMBER OF TIMES TO PROCESS REQUEST.
BLK1|BLK2|BLK3|      BLANK 11×8.5, 8.5×11, OR 16×11 INCH PAPER.
     IPS1|IPS2|IPS3       AS ABOVE, BUT WITH IPSA LOGO.
     TRA1|TRA2            SIZES AS ABOVE, BUT PLASTIC TRANSPARENCY SHEETS.

THE FIRST MENTIONED OF EACH CHOICE IS THE DEFAULT.
ENTER THE DIFFERENT SPECIFICATIONS SEPARATED BY COMMAS, AND END WITH A SLASH.

SPECIFICATIONS :
                 IPS1/
REQ.NO 1502 FILED    1724097 16.41.06   2 SEP 1980
OPTION :
        STOP
```

Figure 16

```
      )LOAD 702 ER586
SAVED  14.33.00 08/29/80


     ∇ LFREPORT1;CPAIR;CARRIERS;REPTYPE;PAX;STS
[1]   ⍝ REPORTS LOAD FACTORS FOR A SPECIFIED CITY PAIR.
[2]    CLEAR
[3]    MONTHLY DATED 1 79
[4]    'CITY PAIR? [E.G. LAX→JFK]'
[5]    →(0=⍴CPAIR←,⍞)/0
[6]    CARRIERS←ERXC CPAIR ⍝ CARRIERS WHICH SERVICE
[7]   ⍝                          THE CITYPAIR
[8]    TITLE 'ER586 SERVICE SEGMENT STATISTICS'
[9]    TITLE 'LOAD FACTORS FOR: ',CPAIR
[10]   LABEL NAME CARRIERS
[11]   PAX←CPAIR ER586 CARRIERS,33 ⍝ PASSENGER COUNTS
[12]   STS←CPAIR ER586 CARRIERS,23 ⍝ AVAILABLE SEATS
[13]   PUT 100 TIMES PAX DIVIDED BY STS ⍝ LOAD FACTORS
[14]   'PLOT, TABLE OR BOTH? [P/T/B]'
[15]   →('P'=RTYPE←1↑⍞)/SPLOT
[16]   DISPLAY ABOVE
[17]  SPLOT:→(~RTYPE∊'PB')/0
[18]   ∆SUPERPLOT 'RESET,ALL/TERM,X1641/'
[19]   ∆SUPERPLOT 'YLABEL,LOAD FACTOR PCT/'
[20]   ∆SUPERPLOT 'SIZE,6 9,HI/TYPE,CRV/'
[21]   ∆SUPERPLOT 'STYLE,SOL,DOT,SDASH,LDASH/'
[22]   'LRTBH' PLOT ABOVE
     ∇

      LFREPORT1
CITY PAIR? [E.G. LAX→JFK]
BOS→DCA
PLOT, TABLE OR BOTH? [P/T/B]
P
SUPERPLOT V3.3 16/8/80
```

Figure 17

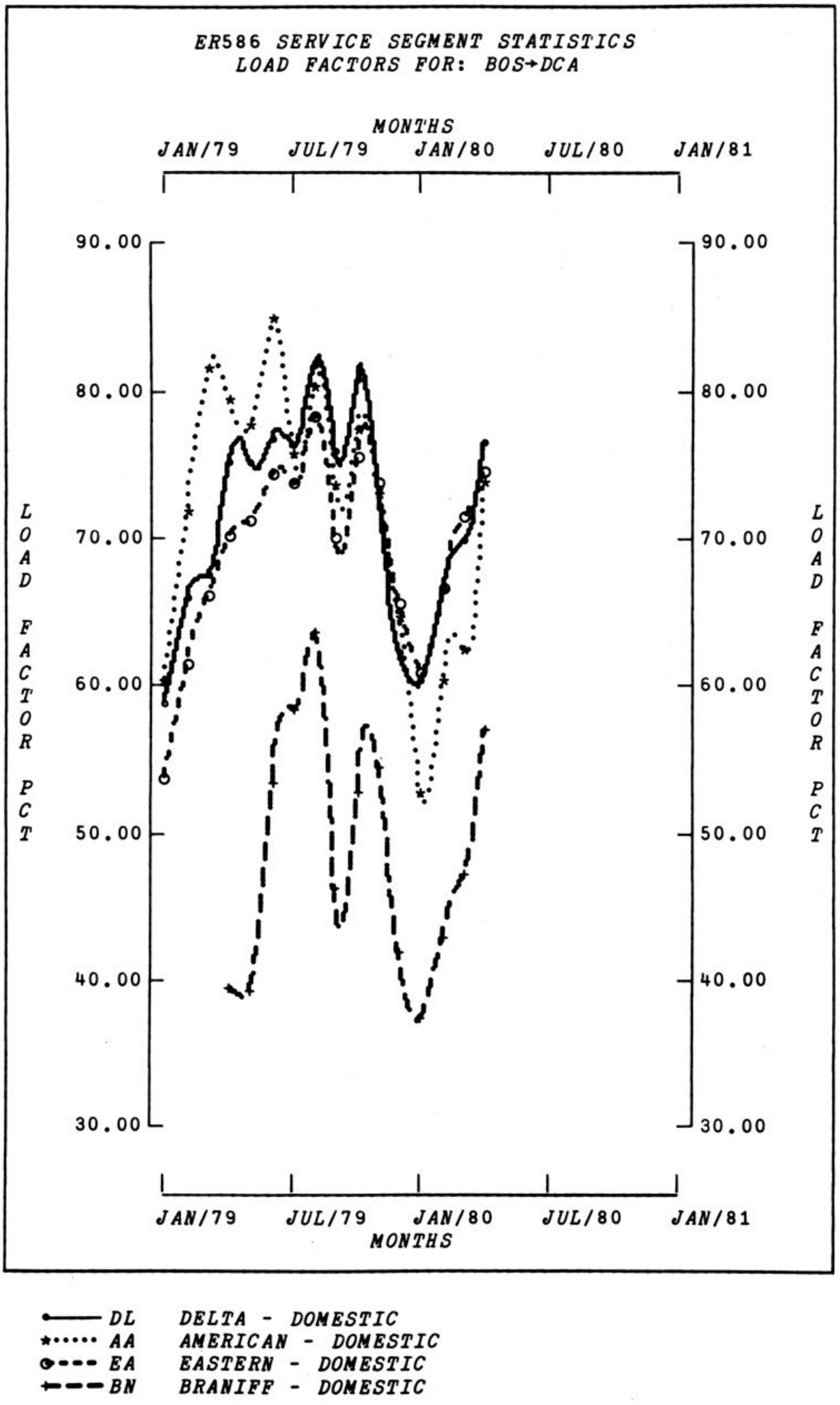
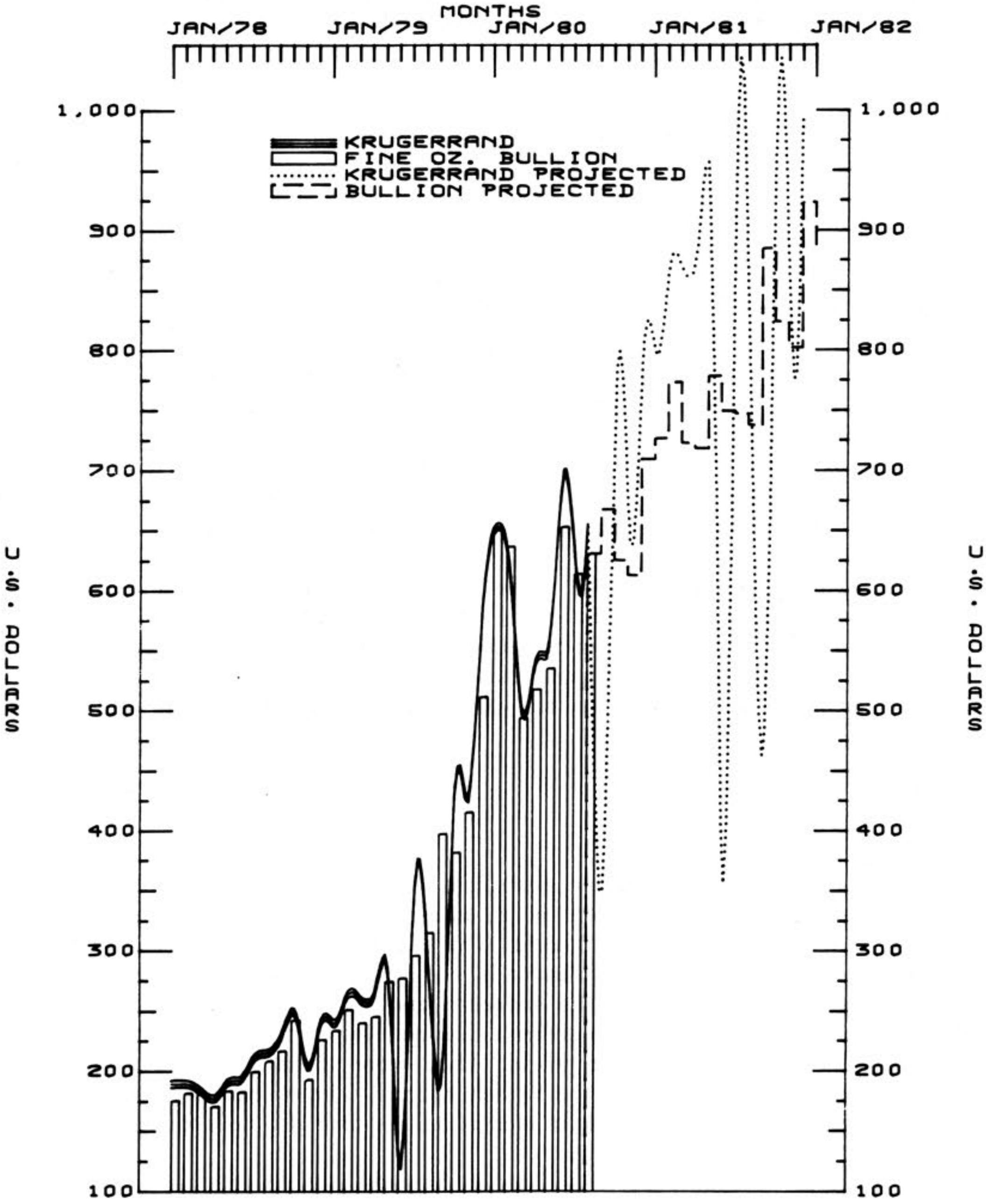


Figure 17 (con't)

GOLD, AVERAGE CLOSING PRICE
ACTUAL AND PROJECTED

(REVISION OF FIGURE 3)



DEVELOPMENT OF A DATA BASED MANAGEMENT INFORMATION SYSTEM FOR PLANNING AND CONTROL

Eric S. Hogg
Corporate Banking Division
Bank of Montreal
Toronto, Ontario

Planning and control within commercial enterprises is a necessity, especially given the frequency of bankruptcies attributable to poor management. Economic success in today's competitive business environment demands among other things, the setting of strategic goals. The definition of these goals into hard and fast financial terms, the communication of these terms to an organization and the reporting of progress are all steps within the process of management by objectives.

Corporate Banking Division of the Bank of Montreal depends on a management information system based on both historical data and financial projections for planning and monitoring its performance information requirements. Before getting into the details of our system, I think it is appropriate to give you a bit of history involved and some background on our structure so that you can understand the context within which our system operates.

Banking — Past to Present

Modern banking began where early capitalism began, namely in the Mediterranean region and spread with the growth of capitalism to southern Germany, the Low Countries, England and the New World. The Medici of Florence were the leading bankers of the fourteenth century, followed by the Fuggers of Augsburg in the fifteenth and early sixteenth centuries. In the sixteenth and seventeenth, the main course of capitalistic development shifted to northern Europe. Antwerp and Amsterdam became the major financial centres of the world. They were surpassed by London in the eighteenth century with the rise of the British Empire. The twentieth century has seen New York assume the dominant role in world banking as it still does today (D. Dillard 1967).

The relationship between capitalism and banking is not slight, as we shall see. The needs of commerce have defined banking activities in the past as they do now. Essentially, this is the reason why the Bank of Montreal set up the Corporate Banking Division.

Historically, banking activities flourished because banks could provide security to merchants. This security was in the form of a fixed exchange rate between deposits and gold — the currency of that period. While coins got lighter and lighter, bank deposits retained their value. So fixed exchange rates, transfer of deposit facilities and safekeeping of wealth allowed merchants to sleep easily at night.

The addition of credit granting capabilities and issuance of bank notes revealed how commerce could be increased through bank loans funded by deposits. This was fully evident in England during the Great Industrial Revolution. Through bank loans, businesses were able to fund the intense capitalization that this period required and which would have been beyond their means if they did not have some access to these pools of capital.

Europe thereby set the stage for the first Canadian Bank which opened its doors in 1817 (D.E. Bond/R.A. Shearer 1972). As Canadian commerce grew, so did Canadian banks. Expanding through branch networks, the banks moved increasingly westward as the needs of people and business demanded.

Originally, Canadian banks were solely devoted to localized populations and firms. However, as railroads were built and cars were introduced, businesses became larger and more specialized. Companies began to deal with more and more branches as their needs and operations increased. In addition, increasing sophistication of financial management activities caused business to demand more and more tailored services.

Recognizing the needs of corporate customers, the Bank of Montreal set up, in 1976, its Corporate Banking Division. With dedicated specialists, the Bank could effectively communicate with customers at senior levels and understand their needs and wants.

The future only holds out more developments in this way, as all banks try to give better levels of service to all their customers.

Corporate Banking — An Overview

Corporate Account Management is the central concept around which the Division is built. The primary objective is to provide large corporations in all fields of enterprise with efficient, professional, dedicated service to meet all their banking needs. To achieve this goal, each corporation is provided with one point of contact for all of its needs; the Corporate Account Manager.

The Account Manager retains total account responsibility for all corporations assigned to him. With the support of a team of experts in credit, cash management services, profitability, pricing and servicing, he is accountable for:

- credit control and negotiations
- non-credit services
- profitability/pricing
- business development
- adequacy of service

In short, he is the focal point for the total client/banking relationship. Decision making authority is housed right in the Corporate Banking shop, and the shortened communication channels ensure prompt, effective decision making.

In actual operation, centralized **Management** is provided from Corporate Banking, while decentralized **Service** is provided by the 1,200+ service branches located throughout Canada. The functional operation is illustrated by the simplified high level organization chart provided (see Chart 1). The Domestic Banking branches provide servicing, with management coming from Corporate Banking Divison. I would like to expand on the integrated operation which is the key to overall effectiveness. In this information flow chart (Chart 2), you will note that detailed transactional information is generated at the branch level, and must flow upward to the ultimate point of accountability — the Account Manager. I'll come back to the flow of information in more detail later, but just now, I believe it's necessary to expand on our organization and how it operates.

The third chart (Chart 3) shows a more detailed picture of how the Corporate Banking Division is structured. The five line groups are split along industry lines, i.e., Natural Resources, Real Estate, Insurance, Commercial and Industrial and Project Financing. Each group contains Account Management specialists. In support of the Account Managers are the staff specialists such as credit analysis, pricing and profitability.

Groups such as these are located in major urban centres adjacent to the Head Office of corporate clients. At present, such groups exist in Montreal, Toronto, Calgary and Vancouver. Of course, our Domestic Banking service branches are located all over Canada, wherever business warrants. As an illustration of the total account relationship, a national retail shoe company has outlets throughout Canada, serviced by our local branches. The Head Office of the firm, located in Toronto, is managed by the Toronto Corporate Banking segment, Commercial and Industrial group.

In considering this example, you will realize that the Corporate Account Manager will require extensive information, of several types, to manage his accounts adequately. Some of the necessary information categories are:

- dynamic daily operation information, e.g., loan/deposit balances;
- historic information for monitoring account operation, e.g., average balances — loans/deposits;
- profit and loss data, e.g., revenues, expenses, volumes; and
- credit analysis information.

Birth of a Reporting System

To fulfill the needs previously outlined, the Bank considered two major alternative approaches. These alternatives were to create separate Corporate Banking Branches to service the Division's clients, or, to employ our existing system of 1,200+ branches to service the accounts, with management being provided by Corporate Banking Division.

You are already aware that we chose the second alternative. I would like to expand on the reasons why we made that decision.

Historically, within the Bank, each branch has been a profit centre, with a detailed funds and revenue plan, and periodic monitoring of actual results to plan. The rewards and punishment system is geared to branch stand-alone profitability, and this has provided adequate motivation.

In· considering the needs of our corporate clients, it was obvious that they required service not just in Toronto or Montreal but in all of their locations spread throughout the country. We could not centralize the local servicing needs of the client's outlets such as the need for cash, coin and other basic services. At the same time, we could not let each of our small branch outlets negotiate pricing with a client independently. The result would have been chaotic pricing and deteriorating client relationships. We were faced with the need to accommodate the branch profit centre concept, while establishing a unified central account management relationship with corporate clients. Our solution was to have the branches continue to **service** these clients, and to establish a central **management** group to handle the accounts. That is the Corporate Banking concept.

Our information systems had to accommodate that decision. The system had to allow branches to maintain balances and to collect revenues for motivational purposes, while providing Account Managers with all the necessary information for centralized management of their assigned account.

Based on the needs previously outlined, a decision was made in 1977 to develop a data base, and I.P. Sharp Associates was awarded the contract. The system was to be designed to allow reliance initially on manual input, with a capability to move in stages to automated updates as the Bank developed further mechanized information extract capabilities. This has, in fact, occurred over the last three years in several stages, and is not yet fully completed. Interestingly, the continued quest for automating input has spun off several valuable new computerized products for the Bank; products which we may not have developed had we not had the basic APL system in place quickly, allowing us to devote time and resources to enhancements. Understandably, this is considered as a positive result of going to a service bureau rather than attempting to build a system internally.

Reporting System Details

I will now deal in some detail with the information gathering and report generation activities associated with the system.

From the previous discussion on information flow, you will realize that the source of data, aside from that provided by client corporations, is at the transaction processing points — i.e., the service branches. It is then necessary to gather, consolidate and report on these widespread operations in a timely and concise manner, to ensure the Account Manager is able to maintain overall control of the operation of his assigned accounts. This was the basic purpose of our Corporate Banking Information System. A system designed to provide our Managers and Executives with the full range of information required for Account Management.

In actual operation, service branches maintain all customers' records for loans, deposits, etc. These records, some automated, some manual, are updated from customers' transactions. In the automated mode, an on-line system for deposits and loans provides real-time accounting and maintains the customers' records, the accounting system, and history files. These are accessed via tape to provide monthly batch updates to our data base. Manual records are updated monthly via branch manual reports which are forwarded to us for input.

Mechanized extracts are made at the detail record level from customers' record files. Pre-processing is done to summarize the detail records, which then update the APL files at a connection level. The types of information accessed can be classified as follows:

Item	Source
Deposit Balances (Hi/Low/Average)	Mech Extract
Loan Balances (Hi/Low/Average)	Mech Extract
Interest Expense	Manual Branch Reports
Interest Revenue	Mech Extract
Sundry Revenue	Manual Branch Reports
Static Data for Control (i.e., names/rates/ authorizations)	Manual Input from Account Managers

In addition, outside of the APL data base, we maintain terminals which can directly access the real-time branch operating data base. These are used to obtain dynamic account information where an Account Manager may wish up-to-date balances or revenue status on specific accounts. Development of this access to meet the dynamic information requirement gave rise to our latest product — i.e., client on-site real-time terminals, located on the desk of the Vice-President, Finance, and used to monitor the client's own accounts without the necessity of contacting a Bank Branch. This exciting spin-off may not have become a reality had time and resources not been freed up by the fact our data base was brought so quickly to operational status.

In addition to the operating information specified, plan and forecast data are maintained at detail levels, and matched against actual performance. The capability is maintained to generate reports in either the Corporate Banking or Domestic (Branch) Banking organizational hierarchies, thus permitting both Divisions to monitor relative performance.

Report output is broken down into four major categories:

- financial planning and reporting of deposits and loans, e.g., balances/rates/ yields/revenue/expense statements

- operating information relating to loans, deposits and revenues, e.g., hi/low/ average balances actual/plan/forecast comparisons

- periodic profitability statements indicating "bottom-line" profits at various levels,

- historical credit analysis information, e.g., authorizations.

A wide assortment of financial statements can be provided for a total picture of financial operations. For example:

- month end loan balances
- month end deposit balances
- changes in month end balances
- mix of month end balances

- yield on average funds
- cost of funds
- major income and expense items
- average funds
- expense statement.

All of these can be prepared at a number of levels to maximize usage, i.e.:

- by client
- by group of related clients (connection)
- by Account Manager's clients
- by Account Management Unit
- by Account Management Group
- Total Division

A similar series of levels can be produced in the Domestic Banking hierarchy. You will realize that most of the levels correspond directly to organizational levels within the Bank, facilitating the provision of required information to various control levels within the organization.

Management Information Requirements

There are essentially two levels of management within our organization, requiring very different classes of information, drawn from the same data base pool.

The primary level is the Account Manager, whose role is that of the day-to-day operating manager, responsible for relatively short-term planning and control of the operations of his assigned accounts. His information needs relate to account level data, on a relatively dynamic cycle (say, up to one month of frequency).

The second level is Senior Management, and although there are sub-levels within this category, we can discuss them as a single block. Senior Management are concerned with strategic planning and longer term operating trends. Their information needs are highly summarized, longer term trends, with details not being necessary.

The Corporate Banking Information System is ideally designed to provide both levels of requirements from one source. The hierarchical reporting capabilities previously outlined allow successive levels of data to be generated from the detailed data, facilitating the examination of trends at successively lower levels as requirements dictate.

Currently, the system monitors some 500 corporations with assets in excess of $9 billion. These assets, entailing more than 1,500 separate records are spread over 270 automated and 30 manual branches.

Let's take a look at the kind of data the system provides to the two levels of management.

The Account Manager is provided with **monthly** reports on the operation of each of his accounts. Necessary control information at the loan/deposit record level is available for him to examine the operation of his accounts to identify exceptions for closer examination if an adverse situation appears to be developing. He can, if desired, use our on-line access system to monitor operations on a day-to-day basis in **Real-Time**. Naturally, this would be reserved for crucial situations in view of the workload entailed. This detailed information is also used in analysing and reviewing authorized credits.

In addition to record level data, the manager is given higher level summation of authorizations, related usage, and yields to assist in maintaining his overall portfolio. All of these are monitored against his annual plan on a monthly cycle, allowing early identification of trends to permit actions for resolution as required.

To assist in his planning function, the Manager can be provided with various levels of historical trend data, which he can use to develop plans and forecasts.

The range of information provided allows a Manager to develop concise, detailed plans, and monitor results against plans on a timely basis, permitting early identification of adverse trends, and allowing him to take effective counteraction on a timely basis. We believe the system provides an effective information base to support the Account Manager in ongoing operations.

At the Senior Management level, the detailed data provided to the Account Managers is summarized by Group and at Total Corporate Banking levels, to provide information for strategic planning. As the level of summation expands, the timeframe also extends. The Senior Executive is less interested in daily or even monthly aberrations. He wants to examine the longer term trend of key items such as funds employed, yields on cost, etc., and, of course, the overall plan becomes his key control mechanism.

Overall portfolio management based on projections of economic developments are an important aspect, as is the assessment of risk entailed in various mixes of deposits and loans, which must be weighed against profitability. The competitive aspects also become important, with categoric market share a key issue dictating strategies and as always it all comes back to bottom line profitability, which is a direct result of astute management of all the above factors.

Again, our Information System, with its ability to summarize at progressively higher levels, is an invaluable tool to the Senior Executive. Quick response to questions generated by high level trend reports is efficiently supported. And, of course, it **is** a closed loop! As trends become evident at the top, Account Managers are required to take specific types of action which achieve adjustments to trends, which takes us back around the loop to daily operating control.

I think you can see that we require a dynamic, responsive system to meet our overall needs. I believe our APL system provides effectively for our requirements.

Unique M.I.S. Features

The information system which I have described to this point has some unique features about it. First, no prototype system existed to meet the information needs of our department. The concept of the division, which oriented its operations to major corporations and their bank activities wherever they might be, was a new one. Clearly the old system of branch accounting and its inherent hierarchy would not satisfy the needs of Corporate Banking. This leads to my second point. The Management Information System which is used in the Division bears little resemblance to the total bank's formal accounting system. The formal accounting system of the Bank of Montreal is branch oriented. Corporate Banking's system is corporate-relationship oriented. While our base data is the product of the former system, the data is stored not by geographic entity but by company and industry groups within management responsibility spheres. The intended integration of other pieces of information unrelated to branch operations such as payroll processing fees will increase the differentiation of the two systems. It is information on the total profitability of doing business with particular connections which we wish to produce. Only a data based management system can achieve this.

The final point which I would like to mention is that our system requires production of both balance sheets and income statements. Not only this, but also that they must interrelate based on the terms and agreements of the credits which the Bank has authorized. Most large firms can produce decision support systems based on income statements. Some can also produce balance sheets. The necessity to closely interrelate the two, however, is definitely unique.

Benefits of APL

The unique qualities which were basic requirements of our system made the choice of APL as our operational language an easy one. We all know how effective it is. Its matrix orientation and manipulative abilities are extremely well suited to business applications. However, there is one point which seems to me to stand out above the rest. This is speed of implementation. Time is a precious commodity, especially in business. APL allows development and enhancements to be done without the long development timeframes of other languages. The same processors also allow modularity. These features allow you to stave off "Future Stock" and keep application packages up-to-date. While it may be contrary to popular opinion, the environment, certainly at the Bank of Montreal, is anything but static.

In spite of high operating costs APL has been, is and will continue to be an effective means by which the Corporate Banking Division can meet the needs of a decision support system.

At this point, you will have been able to achieve a fair understanding of our decision support system. While we might experience difficulties collecting and maintaining our data base, an independent system has proven to be the right way to go for us. Both account management and senior management can be provided with the required level of information needed to support the decisions which they must make. APL has allowed us to do this and to get it done on a timely basis. Our system gives us good control of our operations and provides a framework within which to plan. While we are always looking to make improvements, the result achieved to this point has certainly been satisfying.

Future Possibilities for APL Applications

The area of planning and control for the entire Corporate Banking Division is not the only useful place to employ a data based system to provide information. The area of non-lending services holds out some interesting possibilities for APL applications.

A short definition of non-credit services is as follows. Those services of account management which are not related to loan activity, such as computerized cash management products, securities, foreign exchange, collections, etc., are referred to as non-credit services. These services are provided through the account manager who is supported by a team of specialists able to create specialized packages for corporate customers.

Large corporations with diverse operations are ideal users of these packages. Some typical products combined to suit a customer's needs are:

- funds concentration services
- consolidated balance information
- payroll procession
- account reconciliations
- receivables collection services

The goal is to provide a full range of cash management services to corporate customers priced competitively and relative to the cost of providing the service where and when it is needed.

It is the area of cost estimation for pricing decisions which is the focal point of our interest. Presently, this activity is very time-consuming and labour intensive.

While the bulk of our non-credit services are automated, the point of service for our products remains the field branch locations. So while some volume related data is available on our mechanized system, most of the detail information must be requested from the individual branches. This can amount to pounds of paper if as many as 100 branches are involved. Not to mention the man-hours required to collect, then analyse this information.

The cost of doing business with a corporate customer relates to the number of branch services they use and the number of cheques which they both issue and deposit. Each individual item is recorded, summed and then costed to arrive at the full bank cost. Pretty tedious work.

What we would like to develop is a simplified method of accurately estimating the direct and overhead costs associated with servicing a customer. Through a data based system using mechanized monthly inputs we hope to turn a tedious and infrequent job (due to resource and data collection problems) into a system of monthly monitoring. Such a system would highlight only the exceptions where activities had substantially changed for analysis rather than the present method of periodic review of all existing accounts regardless of whether or not changes had occurred.

Senior management as well as account management could set both overall and individual profit goals based on a stream of reliable information which this system would provide.

However, this account analysis system would have to meet all the same requirements that our reporting system meets. Specifically, we would require speed of set up, flexibility to make changes, timeliness of reports, and data based maintenance capabilities. It will come as no surprise to you that we foresee APL as a very satisfactory vehicle to achieve our goal.

In Conclusion

As you can see, our information systems are continually evolving to meet ever increasing management needs and ever changing organizational structures. Our data based systems must provide the supporting tools to various levels of management to enable them to plan and control effectively. We aim for the future to provide the most complete information possible. The great thing about APL is that the timeframe from germination to completion of a system is thankfully short. It seems in practice that conceptualization is more time consuming than actual implementation.

References

1. D.E. Bond, R.A. Shearer, **The Economics of the Canadian Financial System: Theory, Policy and Institutions**, Prentice-Hall of Canada, Ltd., Scarborough, 1972.

2. D. Dillard, **Economic Development of the North Atlantic Community**, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1967.

CORPORATE COMMUNICATIONS USING THE SHARP APL MAILBOX

Leslie H. Goldsmith
I.P. Sharp Associates Limited
Toronto, Ontario

Abstract

Perhaps more than any other single factor, the advent and growth of electronic mail systems has been markedly changing the appearance and pace of modern offices. Access to an ordinary telephone is today synonymous with access to world wide telecommunication networks. This, coupled with the fact that timely and cost-effective communication is absolutely crucial to any corporation, particularly those which are geographically dispersed, has made electronic mail an ideal mechanism for transferring information.

The SHARP APL Message Processing Facility (Mailbox) was developed ten years ago in early recognition of a growing need for efficient inter-company and intra-company communication. The reasons behind this growth are discussed in the paper. Some of the features of the Mailbox which make it particularly attractive, and some of the human factors involved with using computers to send messages are also addressed. In addition, the paper briefly examines certain design considerations associated with developing the Mailbox, including providing both security and a high degree of message integrity.

1. Introduction

Few technologies are presently receiving as much attention in business and management circles as is the area of communications. Man has been communicating, with varying degrees of success, since the beginning of time. And yet, even today, studies have shown that the single most important factor impeding increased office productivity is communication.

Every company is, to some degree, dependent upon communications to conduct its day-to-day business. In fast-paced business situations, it is vital that a company be confident that the necessary information is being directed to the appropriate individuals as efficiently as possible. Messages which are delayed or not delivered can mean lost business or cause incorrect actions to be exercised in delicate situations. Further, a message delivered late can often cause someone to act upon information which is no longer valid.

"Electronic mail" is a growing field which offers an extremely attractive solution to the problem of providing timely and cost-effective communication, both within a company and outside it. The fundamental concept of sending information as impulses along a wire is not new; indeed, systems employing this basic idea have been in operation since 1844, when the United States government completed the first telegraph line from Washington to Baltimore. The first message transmitted over this telegraph line was, "What hath God wrought?" Now, more than one hundred thirty-five years later, people planning and directing modern offices are asking themselves the same question.

2. The Growth of Electronic Mail

During the last decade, certain factors emerged which together have become the driving forces behind electronic mail. First, electronic technology has made it possible to implement large international electronic mail networks. This means that a user may now communicate with a central computer or another user over a much expanded range. Second, large scale integration technology has made it economically feasible to build low-cost intelligent terminals and communications interfaces. Third, large organizations, or organizations with geographically dispersed personnel, have required enhanced communication mechanisms to effectively manage themselves and to disseminate information in a controlled fashion. This ability to communicate efficiently has had an increasingly high value in our highly competitive business environment.

Since the beginning of the Industrial Revolution, management has focused on the factory, rather than the office, as the place to improve productivity. Indeed, the tremendous strides made in the manufacturing process have contributed heavily to the success of industry. Yet today, there is growing concern about our declining rate of productivity.

Each year, white-collar workers are steadily growing in number and in impact on our society. Peter Drucker, in his book **Management: Tasks, Responsibilities, Practices** [Druc74], supports this view and observes "Increasingly, the central human resources are not manual workers — skilled or unskilled — but knowledge workers: company presidents, but also computer programmers, engineers, medical technologists, hospital administrators, salesmen and cost accountants; teachers and the entire employed educated middle class which has become the center of population gravity in every developed country."

The emergence of the "information society" is represented in the growing concern for professional effectiveness. According to the U.S. Department of Labor, postwar productivity in the manufacturing sector of the economy declined from a 3.3 annual percentage rate during the years 1947-1966, to a 1.5 annual percentage rate during the period 1967-1977. In the same period, wages have risen over 6 percent annually. Add to wages the overhead associated with office workers, and the problem of spiraling administrative costs becomes obvious.

Although there are a variety of causes for the declining rate of productivity growth, lack of progress in the office is clearly a major contributor. While advances in science and technology have steadily improved productivity in the agricultural and industrial sectors of our economy, improvements in the information sector have definitely not kept pace. In fact, offices have remained virtually unchanged for over a century. Electric typewriters are more sophisticated than their manual counterparts were one hundred years ago; microprocessor-driven calculators have replaced noisy mechanical adding machines; and photocopiers have made a lasting appearance. These are certainly the most important innovations in the office, and indeed, practically the only ones.

Why have increases in office productivity not been comparable to those occurring in the factory? The basic lack of technological equipment specifically designed to enhance productivity is the most obvious reason. It has been estimated that the American management community sustains a per capita investment of about $25,000 for each blue-collar worker, and roughly twice that amount for each agricultural worker [Grov79]. This is in sharp contrast with the average capitalization per office worker: estimates range between $2,000 and $4,000 or only 4-8 percent of the investment per worker in agriculture.

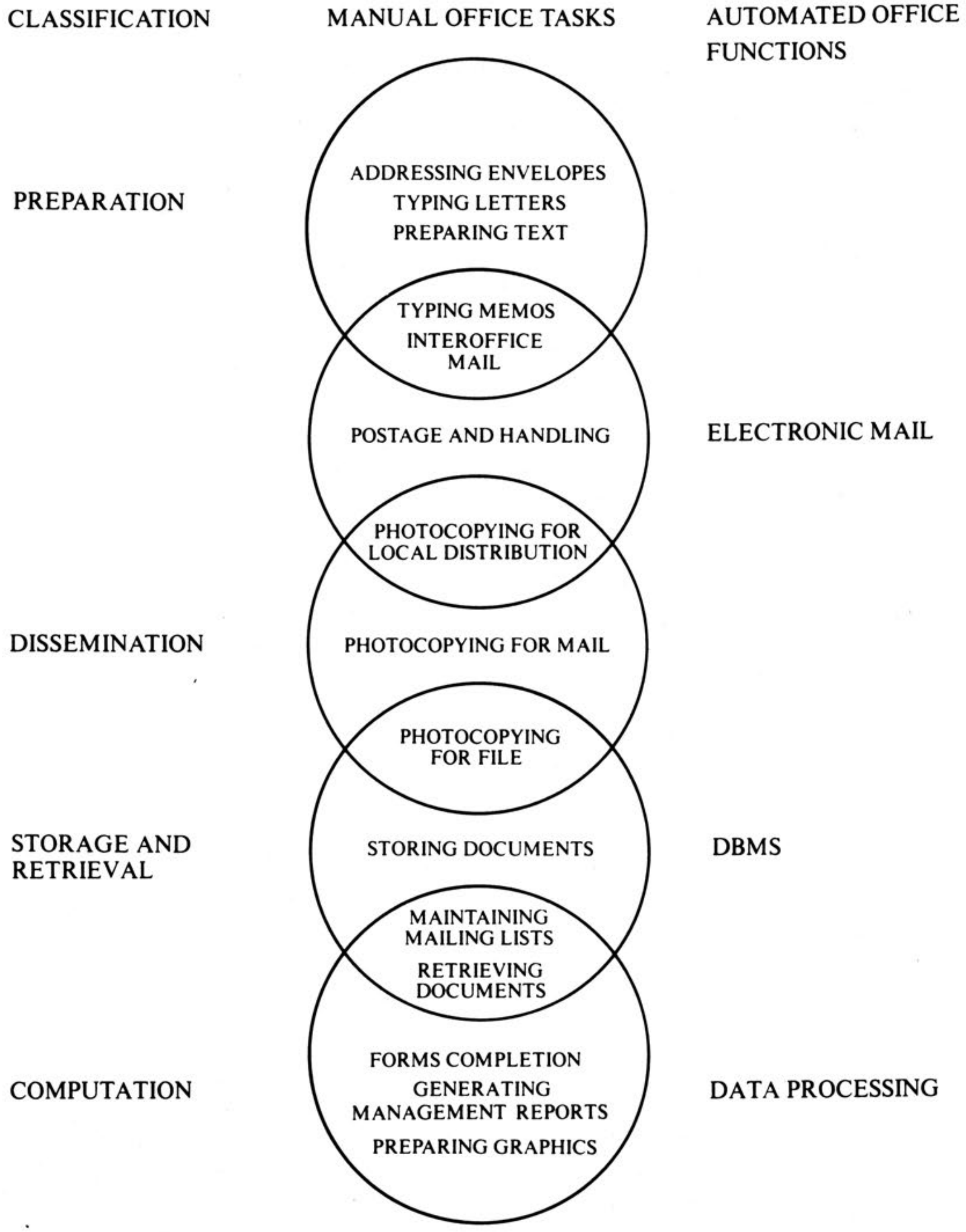
While offices have remained virtually stagnant, the business environment they are endeavouring to service has changed dramatically. Contemporary offices function in a fast-paced, sophisticated world, and the scope of problems confronting businesses today requires adaptability and control far beyond that offered in the typical modern office. To be sure, the need for proper communication channels has never been so great.

Why then are the two most common forms of dispersed corporate communication, the telephone and the postal service, so inadequate? In the United States alone, almost 75 billion pieces of mail are moved annually by the postal service and, of this figure, four out of five pieces are business letters [Gott77]. No data is readily available concerning the number of pieces that are lost, damaged, or pilfered. But that aside, the United States Postal Service, and their marginally less-efficient Canadian counterparts, are hard-pressed to cope with the current level of postal traffic. Mail simply doesn't move fast enough or reliably enough in many cases. In fact, it often seems that first class mail moves slower than it did in the middle of the last century by Pony Express! And even when it is moving, both postal services have the uncanny ability to go on strike just at the time they can disrupt the largest volume of mail.

Mail service difficulties are by no means relegated to the government postal services. Many companies are experiencing similar turnaround problems with their own intra-company mail service. Research into one large organization, Exxon Corporation, reveals that in-house mail spends one percent of its time being originated, one percent being copied, three percent being typed, twenty-one percent sitting in incoming/outgoing bins at various points during its journey, and seventy-four percent in the delivery system [OEM80]. A full forty-five percent of the internal Exxon mail spends three days getting to its destination — days that could be critical to decision making. These figures are typical for large corporations, and almost certainly illuminate what the future might hold for smaller corporations which are expanding at a rapid rate.

In anticipation of these continued problems, many companies have begun investing in alternative forms of moving letters and packages vital to successful operation. Parcel and courier services are the obvious choice for packages. Electronic mail is the obvious choice for letters and memos. As shown in Figure 1 (after [Mier80]), electronic mail can be viewed as a superset of not only the postal system, but also interoffice mail handling, memoranda, and mail reproduction. Electronic mail obviates the need for reproducing mail because each recipient receives his own personal copy.

In contrast to the postal system, the inefficiency of the telephone stems mainly from its inherent requirement that individuals be simultaneously in a position to converse with each other. The most striking cases of this occur with people attempting to communicate across time zones, but the fundamental problem is much more general and is one most everyone has encountered. For example: Mr. Allen needs to discuss a problem with Mr. Burger. Allen calls Burger, who is out of the office, so someone takes a message. When Burger returns the call, Allen is tied up on another line, and so Burger leaves a message. Allen calls again, but now Burger is in a meeting. When the meeting ends, it's already almost five o'clock. Burger pessimistically calls back, and is unsurprised to learn that Allen has already left for the day to go to his favourite pub. This exercise in futility has become one of the most frustrating games in business — and it can go on and on, sometimes for days. Studies have shown that, on the average, the caller fails to reach the person being called on the first attempt in 28 percent of all business telephone calls [Pott77]. The same type of thing can occur when trying to get together for a face-to-face meeting. Communication merry-go-rounds can cost valuable time in crucial situations.



Manual Office Task and Their Automated Counterparts

Figure 1

The telephone has other serious inefficiencies as well. Phone calls often come at inconvenient times, or interrupt a train of thought. The recipient of the call might be tired, or otherwise in an unreceptive frame of mind when the call is received. This can result in pretentious displays, where the recipient feigns interest, or even misunderstandings. After all, it is unlikely that the caller (who is probably in a reasonable frame of mind if he is making the call) will of his own volition attribute the recipient's attitude and tone to an off-moment or an off-day. With the Mailbox, the recipient can choose when and where he wants to receive his communications, and can adjust the timing to when he is best able to cope with them.

Telephone communication also has the disadvantage of being verbal and impermanent. At the end of the call, there is no written record of what conclusions or decisions were reached, or of those that were not. And of course, no telephone call would be complete without the social amenities and convivialities that accompany verbal exchanges.

The inefficiencies of the telephone are supported by a study conducted by Fortune magazine, in which managers consistently cited the telephone as the highest time-waster [Rowa78]. The situation is indeed ironic, for the telephone is undoubtedly the most widespread method of communication in common use today.

3. The Mailbox System

The SHARP APL Message Processing Facility, known as the Mailbox, is an advanced electronic mail system that is available throughout I.P. Sharp's international time-sharing network. The Mailbox is secure, and provides an efficient means of electronically moving information between people, wherever they may be, rather than between geographic locations. By interacting directly with a central computer via a remote terminal, the user can bypass all the costly and time-consuming steps normally associated with the preparation of a letter or other written communication. The same efficiency and permanence of the Mailbox, as well as its positive timing-related impacts, makes it an attractive substitute for other forms of communication as well.

Each Mailbox user has access to a terminal. Since the terminal serves only as a link to the computer, the same one needn't be used all the time. For example, while travelling, it is often convenient to carry a portable terminal to stay in touch. The fact that the actual location of the terminal is irrelevant, is one of the factors that makes electronic mail so unique.

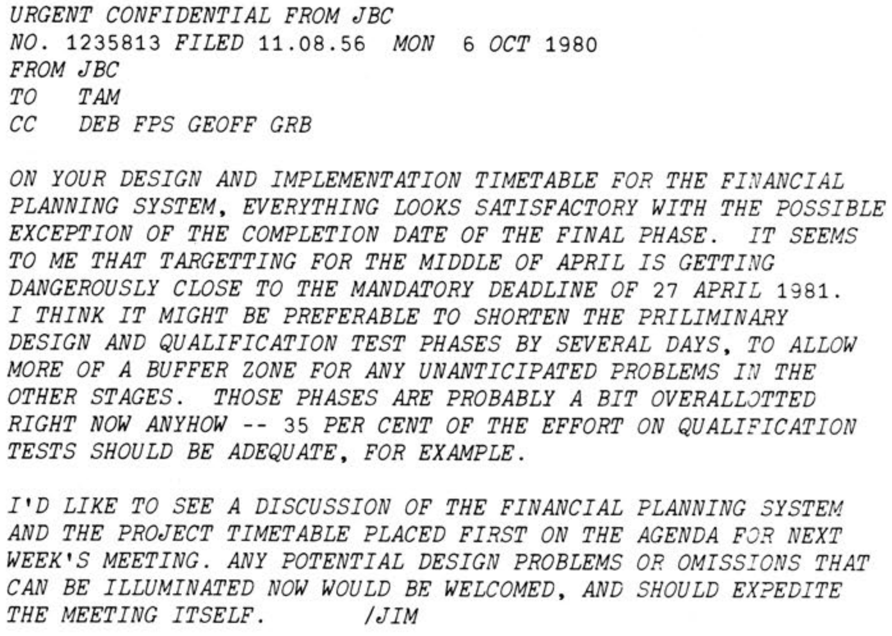
Associated with each member of the Mailbox is a unique **address code**. The address code serves to identify the senders and recipients of messages filed through the Mailbox, and is usually the person's initials or a short form of his name. William Shakespeare might have the address code *WMS* or *BILL*, for example.

Any number of individuals can be assembled into a Mailbox **group**. A group contains the address codes of each of its members, who usually share a common interest or are associated with the same company or organization. For example, the group *MKTG* might contain all people involved in marketing. A message addressed to a group is sent to everyone in the group, as if each address code had been typed explicitly. Thus, it is as easy to send a message to a single individual as it is to address a large distribution list.

A message can be sent to one or more individuals or groups, and also "carbon-copied" (*CC*'d) to others. An additional distribution list feature lets the user carbon-copy people, without making this fact apparent to any of the message's other recipients. Before a message is sent, it can be optionally classified as *CONFIDENTIAL*, *PERSONAL*, or *PERSONAL AND CONFIDENTIAL*. These external classifications are largely a matter of convention between sender and recipient, because the Mailbox always carefully guards each message regardless of its privacy level.

A message can also be marked *URGENT* and/or *REGISTERED* by its sender. Urgent messages print before non-urgent ones when the recipient reads his mail. Registered messages cause a confirmation of receipt to be reported back to the sender each time the message is received. The *REGISTERED* classification is not made apparent to any recipients of the message.

Figure 2 illustrates a sample Mailbox message. When a message is sent, the Mailbox automatically assigns a unique message number to it, and records the time and date it was filed. Along with the distribution list and external classification set by the sender, this information is assembled into a message "header". The text of the message follows the header, and may be anything from a simple memo to an elaborately-formatted document or report.



```
URGENT CONFIDENTIAL FROM JBC
NO. 1235813 FILED 11.08.56  MON  6 OCT 1980
FROM JBC
TO   TAM
CC   DEB FPS GEOFF GRB

ON YOUR DESIGN AND IMPLEMENTATION TIMETABLE FOR THE FINANCIAL
PLANNING SYSTEM, EVERYTHING LOOKS SATISFACTORY WITH THE POSSIBLE
EXCEPTION OF THE COMPLETION DATE OF THE FINAL PHASE.  IT SEEMS
TO ME THAT TARGETTING FOR THE MIDDLE OF APRIL IS GETTING
DANGEROUSLY CLOSE TO THE MANDATORY DEADLINE OF 27 APRIL 1981.
I THINK IT MIGHT BE PREFERABLE TO SHORTEN THE PRILIMINARY
DESIGN AND QUALIFICATION TEST PHASES BY SEVERAL DAYS, TO ALLOW
MORE OF A BUFFER ZONE FOR ANY UNANTICIPATED PROBLEMS IN THE
OTHER STAGES.  THOSE PHASES ARE PROBABLY A BIT OVERALLOTTED
RIGHT NOW ANYHOW -- 35 PER CENT OF THE EFFORT ON QUALIFICATION
TESTS SHOULD BE ADEQUATE, FOR EXAMPLE.

I'D LIKE TO SEE A DISCUSSION OF THE FINANCIAL PLANNING SYSTEM
AND THE PROJECT TIMETABLE PLACED FIRST ON THE AGENDA FOR NEXT
WEEK'S MEETING. ANY POTENTIAL DESIGN PROBLEMS OR OMISSIONS THAT
CAN BE ILLUMINATED NOW WOULD BE WELCOMED, AND SHOULD EXPEDITE
THE MEETING ITSELF.         /JIM
```

Sample Mailbox Message

Figure 2

Learning to use the Mailbox is simple. In fact, most users become comfortable with its main facilities within a matter of hours. Mailbox commands are generally English words that are easy to remember because of their suggestive meanings. For example, to determine information about incoming messages that have not yet been received, type *UNREAD*. The Mailbox will respond with the serial number of each pending message, along with its sender, privacy level, and an indication of whether the user was included in the *TO*, *CC*, *BCC* class of the message header. *UNREAD* also tells the user about outstanding messages that he has sent, and which recipients they are still active for.

To get a more condensed report of pending mail, type *PREVIEW*. The Mailbox will respond with the number of pending messages, their senders, and a count of those that are classified as *URGENT*, *CONFIDENTIAL*, *PERSONAL*, and so on.

To read some or all pending messages, just enter the command *PRINT*. If any messages are classified higher than nonconfidential, the Mailbox will ask for the highest privacy level it should display, and will automatically bypass messages above that level. Messages that the user chose to bypass can be printed at any time just by entering the *PRINT* command again. This enables an individual to read all confidential mail, but perhaps save personal messages until circumstances permit them to be displayed in privacy. Further, because Mailbox messages can be manipulated as data, it is possible to involve them in auxiliary operations such as filing, cross-referencing, or retrieving.

Sending mail is also easy. The user types the word *SEND*, and the Mailbox responds by prompting for the address codes of the recipients of the message. When this information has been entered, the Mailbox prompts for the text of the message, which can be of arbitrary length and complexity. After the user signals the end of the message text, he simply gives the command to send the message. The message is filed, and its message number and timestamp are printed; it is then **immediately** available to all of its recipients, regardless of their geographic locations.

Before sending the message, the user can also choose to alter any aspect of it, including the distribution list, privacy level, or the message text itself. Another simple command will cancel the message being prepared and start over from scratch. Even after it has been sent, the sender can still retract the message (at once making it unavailable to its recipients), and then either modify and resend it, or else discard it.

The Mailbox also provides a forwarding facility, so that messages which can be better dealt with by someone not on the original distribution list may be passed on to that person, with one simple command. The user may also extend the message he is forwarding by adding his own commentary or preface to it. Further, if the message is classified as *CONFIDENTIAL* or *PERSONAL*, then the Mailbox automatically notifies the original sender that the message is being forwarded.

Another Mailbox command, *WHOIS*, lets an individual perform a variety of inquiries about himself or other members of the Mailbox. For instance, it is possible to ascertain the address code of another user, the identity of a user with a particular address code, the composition of a group, or the groups to which any individual belongs. All of the facilities of the Mailbox are described in detail in [Gold80].

Using the Mailbox, a person need never feel tied to an office as the only source of information regarding recent developments, and he need never feel out of touch when he can't make it in to work. If the user is on vacation, or otherwise expects not to be reading his mail regularly, he can just send a special Mailbox message to that effect (possibly including who to contact in case of difficulty). Then, anyone who sends the person a message will be automatically notified of his unavailability. When he returns, the accumulated messages will provide a complete account of the activities that transpired during his absence.

The Mailbox user who is away from the office on business can enjoy the benefits of electronic mail wherever he happens to be, and not constantly have to worry about checking in with his office. Consider, for example, the somewhat exaggerated case of the corporate professional who begins a hectic week-long business trip. To him, being out of town is synonymous with being out of touch, and he has no recourse but to attempt to contact his office at least once a day. Even as his plane leaves the runway, calls begin pouring in for him, and message slips begin piling up. But what follows is a series of frustrating attempts by him to reach his office. He may, for instance, try to temporarily escape from one of his many scheduled meetings. However, this can be awkward, or even impolite. He may instead try to call the office during lunch — but where is his secretary? She is out at lunch also, of course. Finally, between afternoon appointments, he spots his chance to catch up on what he's missed. He eventually finds a pay phone to call from, only to discover that he has no change. At the hotel room that evening, our traveller has a lot of time on his hands. Everyone has long since left the office, and there is little to do other than sit back and watch sit-com reruns. A week later, the weary businessman returns home only to find that, despite his valiant efforts, a stack of yellow slips and numerous impatient clients await him. He can now settle back and face the two days' worth of catching up that are in front of him.

With the Mailbox, traumatic experiences such as this are avoided. Today's lightweight portable terminals make it possible to extend the realm of office communication to just about any place, be it home, a hotel room, or even a customer's office. As long as there is a telephone and an electrical outlet around, efficient communication can occur at any time with people who don't know where you are or where you will be — and vice versa.

4. Human Design Considerations

Interactive computing may be loosely defined as an information systems capability which allows both specialists and novices to interact with the computer in a responsive, conversational manner. In order for this to work, the system must combine ease of use and functionality in an environment in which it is comfortable for people to interact. The area of man-machine interfacing plays heavily in this, as it has a major impact on how receptive people will be toward using a particular piece of software. If the software is inflexible or has quirks that cannot be circumvented, there is a strong risk that people will shy away from using it, or be wary when they have to.

The manner in which a piece of software communicates with the end user can be broken down into at least two categories: **procedural** considerations and **syntactical** considerations [Mart73]. Procedurally, the Mailbox was designed for people with or without any computer background. It is both easy to learn and easy to use. For example, the message sending process is composed of a logical progression of steps which, once learned, become subconsciously anticipatory in nature. Several other features combine to make the process simple and straightforward. Where multiple option selections are possible, responses may be combined and placed on a single line. This results in faster dialogue sequences with fewer inputs. In fact, one can even specify the distribution list, privacy level, and text of a message, and send the message, all in one single input line. This is performed most frequently in cases where the text of the message to be transmitted has been previously composed (for example, it might be the output of some earlier phase of processing). In addition, when composing a message, it is always possible to replay all information that has been specified and change any that is not correct.

On syntactic considerations, the Mailbox uses simple prompting schemes which are consistent in format. For example, all prompts ending in "*?*" are questions which require a "*YES*" or "*NO*" response. User responses are typically terse, and may be abbreviated if desired. Also, a user can get a help message at any prompt, just by pressing the RETURN key. (The dialogue structure was designed to minimize the likelihood of help being needed, but occasionally the user might become confused or forget the name of the command he wishes to select.) In addition, the Mailbox will always respond to an entry, so the user knows that his input was either processed successfully or else rejected — and, if the latter, precisely what was incorrect.

The conversational programs provided by the Mailbox serve the vast majority of its users. However, some sophisticated Mailbox members prefer custom interfaces to the message sending and printing routines, and to the inquiry capabilities. The Mailbox therefore also provides a number of primitives to perform base-level functions, such as reading or sending a single message. Users can then write their own interfaces to these primitives, to give the Mailbox the profile they desire. A facility known as *⎕PROF* further provides a mechanism whereby the Mailbox can remember the user's profile, and invoke it automatically each time he accesses the Mailbox. This modularity and flexibility has had very positive effects on user satisfaction. It has also made it straightforward to integrate the Mailbox into other systems which require the ability to automatically transmit status or textual messages to people.

The general conversational programs and the more specific primitive functions combine to make the Mailbox attractive to widely differing classes of users. The system is robust and resilient against errors. Designed to be "bullet-proof", every possible precaution was taken to ensure that a user cannot accidentally or intentionally cause the Mailbox to fail or act in a disadvantageous manner.

5. Technical Aspects of the Mailbox

The Mailbox is composed of a variety of programs for sending, receiving, and disposing of mail, for making inquiries on messages, and for making inquiries on other members of the Mailbox. The system consists of two workspaces and a single APL file. The user facilities mentioned reside in one workspace, which is accessed by people whenever they want to use the Mailbox. The other workspace is used only by the Mailbox stewards, who are responsible for enrolling, changing, and deleting enrollees, altering group membership, spooling and purging messages, and performing a variety of other maintenance activities. The file is a shared storage medium which contains Mailbox directories, status and control information, and the message database.

The development of the Mailbox involved meeting some fundamental design goals. Perhaps the most important of these was the goal of security. The system had to be secure, and maintain a high degree of message integrity at all times. Functionally, the main effect of this is that it is not possible for a user to read another user's mail: all messages are carefully protected, even from the Mailbox stewards. Furthermore, the failsafe file design ensures that the Mailbox remains completely intact in the event of a system crash or hardware failure. The package is well-guarded against both malicious and accidental attempts to compromise its integrity.

Program efficiency was another important design goal. This was complicated by the fact that the Mailbox originally had to run in 48K workspaces, so it could not enjoy the luxury of using storage-intensive algorithms to help achieve efficiency. The time-space trade-off was resolved by using complex file data structures and highly-efficient APL algorithms to manipulate them. In addition, algorithms are sometimes selected dynamically in cases where the most efficient solution is dependent upon data or other non-fixed parameters.

Because the Mailbox was intended to be used by both experienced and untrained users, it was designed with a minimal number of "rules" to remember and two levels of input prompting. Advanced facilities such as Mailbox profiles and non-interactive conferencing are available for the more sophisticated users, but individuals who do not wish to use them need not even concern themselves with knowing about the features. The Mailbox is also fully documented in the user's manual, and a supplementary reference card.

The Mailbox design goals were considered in isolation of implementation difficulties. As a result, the requirements to fulfill these goals fell into two categories: those that could be met with facilities already available, and those that could not. Necessary facilities that were already available pertained to the system environment, notably the large time-sharing network, the SHARP APL File System, and the APL language itself. Required new facilities were not quite as straightforward, and occasionally resulted in modifications to the APL system. The potentially conflicting goals of security and efficiency, for example, led to the ability to "seal" a package and mask the names within it. The resulting encapsulation enabled one to make provable assertions that would have been difficult or impossible to sustain otherwise.

To keep the space consumed by the Mailbox to a minimum, sophisticated program compression techniques, now widely used to maintain other systems as well, were developed. One of the problems that program compression was designed to solve relates to comments within functions. Because APL is interpretive rather than compilation-driven, the image of a program that it executes is almost identical to what the user typed in. In particular, comments, which are normally removed during source-to-object code compilation in other languages, remain intact in running APL programs because there is no notion of object code. To reclaim this space, most comments are removed from the Mailbox prior to its installation. This results in an enormous space saving (over 65 percent), and a small reduction in program execution time as well. Other program optimizations that are performed further reduce the space requirements and increase efficiency.

A number of interesting problems arose during the internal design and implementation of the Mailbox. Many of these were direct results of the specifications and design goals of the system. A few general and specific problems are mentioned in the remainder of this section.

The installation of a new Mailbox, and the reinstallation of a later version of an extant one, had to be done cleanly and in such a way as to preserve the security of the Mailbox at all times. It was also important that people using the Mailbox at the time it was being reinstalled experience no interruption in service, nor indeed any indication that an installation was in progress. Users with privately-saved copies of the Mailbox are, however, automatically notified if the version of the Mailbox they are using has been superseded by a later release.

To protect the Mailbox against software, hardware, or malicious user error, an additional level of validity checking, called "paranoia checks" [Gold78], is performed. Paranoia checks are an extension of preliminary validation checks and semantic plausibility checks, both of which may legitimately encounter invalid or improper data. What sets paranoia checks apart is that they theoretically deal only with prevalidated data, and thus should never fail. The Mailbox uses frequent paranoia checks to guard against possible user error or integrity exposure, and does not assume that a single level of nonredundant checks is adequate to ensure the validity of sensitive operations. Checks that fail are immediately logged for perusal by a Mailbox steward, and the concomitant operation aborted.

Because many users can be accessing the Mailbox simultaneously, multi-user interlocks were necessary to ensure that no unsynchronized file updates could occur in critical program sections. System-level queuing mechanisms provide the basic facilities necessary to properly sequence file updates and accesses; however, the Mailbox itself had to ensure no inconsistent data could result from updates that were aborted (through user interruption or system crash), or that were interrupted and then resumed. Two copies of each directory are used to guarantee that hardware failure or interruption during an update does not damage the Mailbox file, or lose any information within it. To avoid difficulties associated with restarting after an interruption, a general restart capability, called *GOON* (for "go on"), is provided. *GOON* always computes the correct restart point in whatever program is suspended. Because of critical section complications, this is frequently not the line number on which the interruption occurred. Implementing *GOON* required giving careful consideration to program topology, always making sure there **was** some point to restart at successfully.

Mailbox stewarding, such as the addition or deletion of enrollees or groups, had to be able to occur asynchronously to normal Mailbox usage, and transparently to it as well. To achieve this, a special signalling mechanism between the steward and Mailbox workspaces is used to cause all active Mailbox workspaces to be reinitialized whenever a steward has changed something. The reinitialization procedure happens invisibly to the Mailbox user. In addition, special steward-steward interlocks guard against problems that could arise from simultaneously-active stewards.

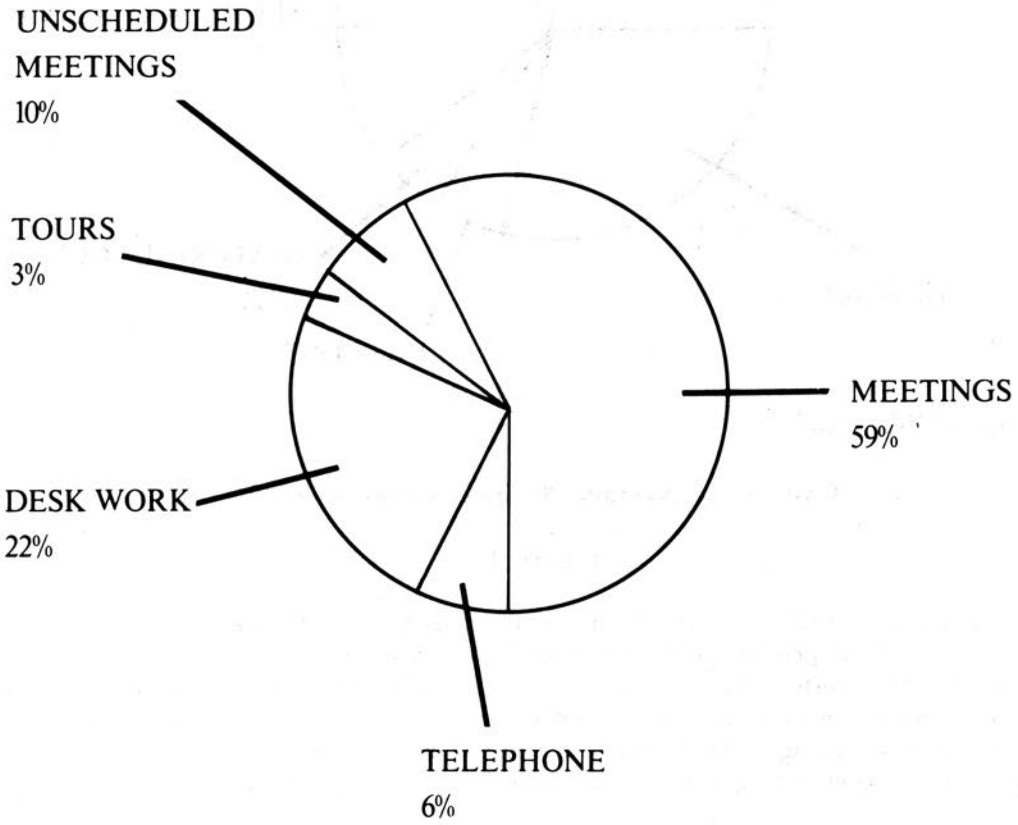
The requirements to handle a large number of enrollees and a large volume of mail (see the appendix to this paper for statistics) suggested the use of data compression techniques in both the file and the workspace. Further, to ensure continuous, smooth performance of the Mailbox, automatic "garbage collection" and capacity expansion facilities were built in. In fact, once a Mailbox has been installed, it requires no handholding at all to keep it running. In the event a problem should arise — or to make sure that none has — a Mailbox self-diagnosing capability can be run by the steward at any time, to precisely pinpoint an error or inconsistency in the system.

6. Economic and Other Benefits of the Mailbox

To analyze the economic payoffs offered by the Mailbox, it is necessary to first examine how it fits into the overall organizational picture. The decline in the rate of productivity growth of private business firms during recent years confirms the need to find ways to increase productivity in the office. Further, the size of the labour force which functions as information workers is expected to at least double during the ten year period beginning 1975, rising from approximately 20 percent of the total work force to over 40 percent. Combining this with the inflationary rise in the basic cost of labour leads to the conclusion that the cost of information processing by people in offices will increase by a factor of more than four. Electronic mail is a fundamental way in which this cost can be cut.

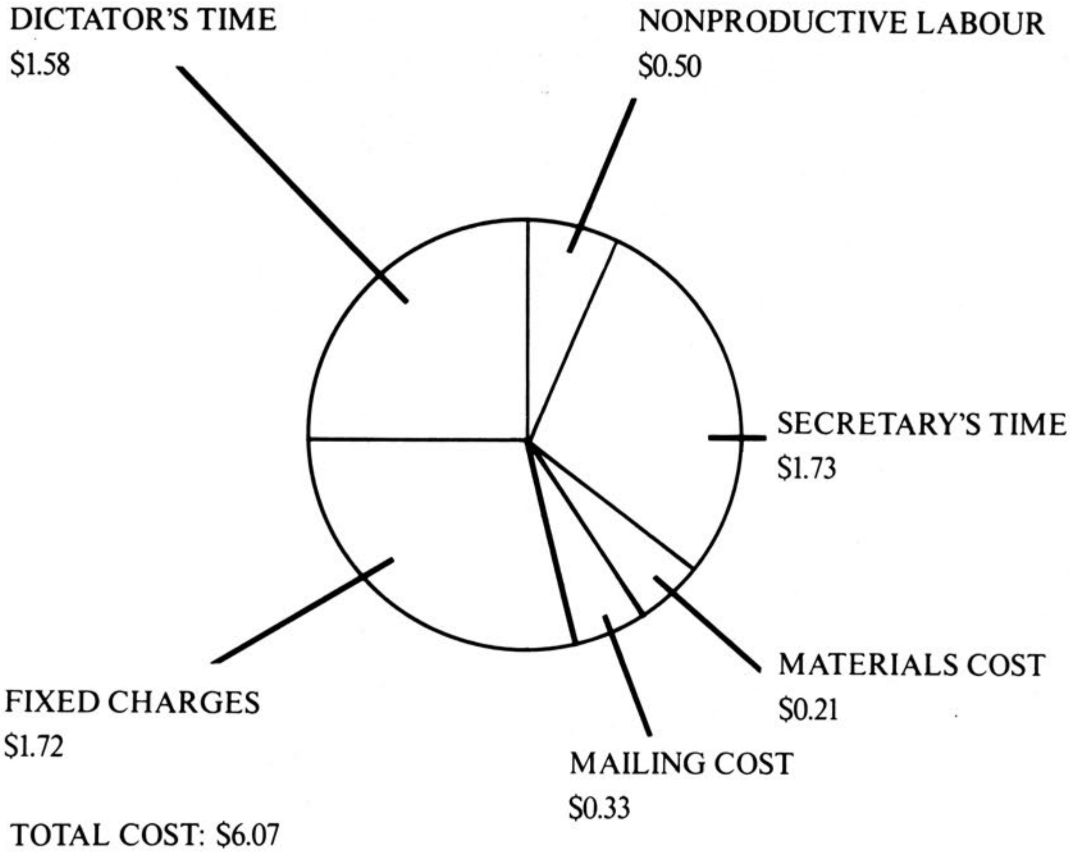
Non-clerical labour costs amount to 66 percent of the total labour costs for white-collar workers [Hark78], so this area is a natural one to examine for the greatest economic leverage. Non-clerical personnel can be divided into essentially two categories, managers and professionals. Managerial activities have been summarized by several persons, and all results are in general agreement. In one of the most interesting and thorough of these studies, Mintzberg [Mint73] concluded that the typical manager's activities can be broken down into five generic categories (see Figure 3). Based on this managerial work distribution, oral communication (meetings and phone calls) amounts to 75 percent of managerial time. If one adds to this the portion of desk work that is involved with writing letters and memos, we find that nearly 95 percent of what a manager does is spent in communication of one sort or another.

Non-managerial professionals tend to spend slightly less time in communication, as one might anticipate. Studies by Bair [Bair74] and a composite study by Panko [Pank76] indicate that professionals spend 63 percent of their time in communication.



Distribution of Managerial Work

Figure 3



Cost of the Average Business Letter in 1980

Figure 4

Since managerial and professional communication gains have such high economic benefits, even small percentage increases in labour and cost savings can have a high impact. But most costs continue to rise at double-digit rates. The cost to dictate and mail the average business letter, for example, increased from just over $3 in 1970 to $6.07 in 1980, according to the Dartnell Institute of Business Research. The breakdown of the present expense to generate a business letter is shown in Figure 4.

Electronic mail offers an extremely viable and cost-effective alternative to escalating communications and labour costs, and decreasing productivity in the information sector. Analysis of the work-time expended in conventional communication compared with electronic mail shows a typical saving of nearly two hours per day [Uhli79] on an activity that already consumed six hours per day (75 percent of non-clerical time). This amounts to a 33 percent improvement in direct communication labour costs. The average hourly wage for non-clerical personnel is presently over $12 (based on Konkel [Konk76] and current inflation rates). Typical overhead and other related expenditures triple this figure to $36 per hour. On a day-to-day basis, this simple analysis shows a potential cost saving of approximately $72 per day for each managerial or professional employee, or $18,000 per year.

To put this into perspective, it costs roughly $1.00 to send a message of about 400 characters in the Mailbox. (The cost is independent of the number of recipients of the message.) Message reception cost is proportional to the number of messages actually received, and is roughly $0.40 plus $0.10 per message. (These figures are based on CPU time consumed to send and receive mail in a private Mailbox containing 800 members.) In addition to this, there are initialization overhead costs of approximately $0.40 when the Mailbox is first accessed in a session. During a typical hour, the average user will send one Mailbox message and receive two. Assuming new sessions each hour, an hour's worth of communication would cost $2.00 in CPU time. Since editing or reformatting a message can increase the cost of sending, we shall assume a nominal cost of $3.00 per hour. This compares extremely favourably with other electronic mail or Telex/TWX-based systems (see, for example, [Barn78] and [Blea80]), particularly since the cost of sending messages in such systems is often proportional to the size and geographical distribution of the recipient list.

Using the hourly rate of $3.00 for computer communications, the daily cost would be $18 based on six hours of communication per day. Taking into account the projected minimum cost saving of $72 per day (two hours of non-clerical labour), the resultant overall savings from this brief analysis is $54 per non-clerical worker per day.

But cost effectiveness is not the whole picture. The Mailbox offers other advantages as well, both direct and indirect. Spatial and time difference factors of communication are minimized. On-line writing is accelerated; an author can compose at a much higher pace because he subconciously knows his wording can be modified using the immediately-available text editing facilities. The increased efficiency of communication increases a manager's span of control. For example, in one study conducted at Citibank [Uhli79], the span of control was observed to increase by 20 to 30 percent. This in turn can have a positive impact on managerial effectiveness.

The collaboration of groups of people is accelerated by the speed of communicating, in terms of both initial distribution and response feedback. Conversations can proceed more effectively, and with a multiplicity of contributions, without the need for physical collocation. The controlled pace which typifies collaborative discussions makes it feasible to increase groups from the classical limit of 8 to 10 participants, to upwards of 50 to 100 persons. Further, regardless of the size of the group, electronic mail reduces the inability or reticence of certain group members to participate in a discussion because of personality differences or shyness.

The use of the Mailbox results in changed communication modes. There is a lessened need to schedule meetings. There is also a marked decrease in the use of conventional mail, and in the use of the telephone — for both local and long distance calls. In addition, the decrease in interruptions which results from the use of electronic mail has the positive effect that working days are no longer "interrupt driven".

The Mailbox also brings flexibility in working hours and in work location. The working day and week are both extended arbitrarily due to the portability of terminals, which can be used wherever there is a telephone. This allows people to work creatively when they feel creative — even if this happens to be on a Saturday. Indeed, one survey conducted by the Yankee Group showed that 20 percent of electronic mail usage occurred outside normal business hours. Another timing-related impact is the instant availability of information. The computer can be used to drive a retrieval system which can store messages (or other documents) for an unlimited period of time. Typical systems permit retrieval based on any number of complex criteria including sender, recipients, date sent, keywords, text searching, and so on.

7. Conclusions

The Mailbox provides a facility whereby the right information can get to the right people, wherever they may be geographically, in a timely and cost-effective manner. With the declining rate of office productivity growth, the spiraling cost of non-clerical labour, and the rapidly decreasing cost of computers and information networks, the time has never been more ripe for the application of electronic mail to corporate communication.

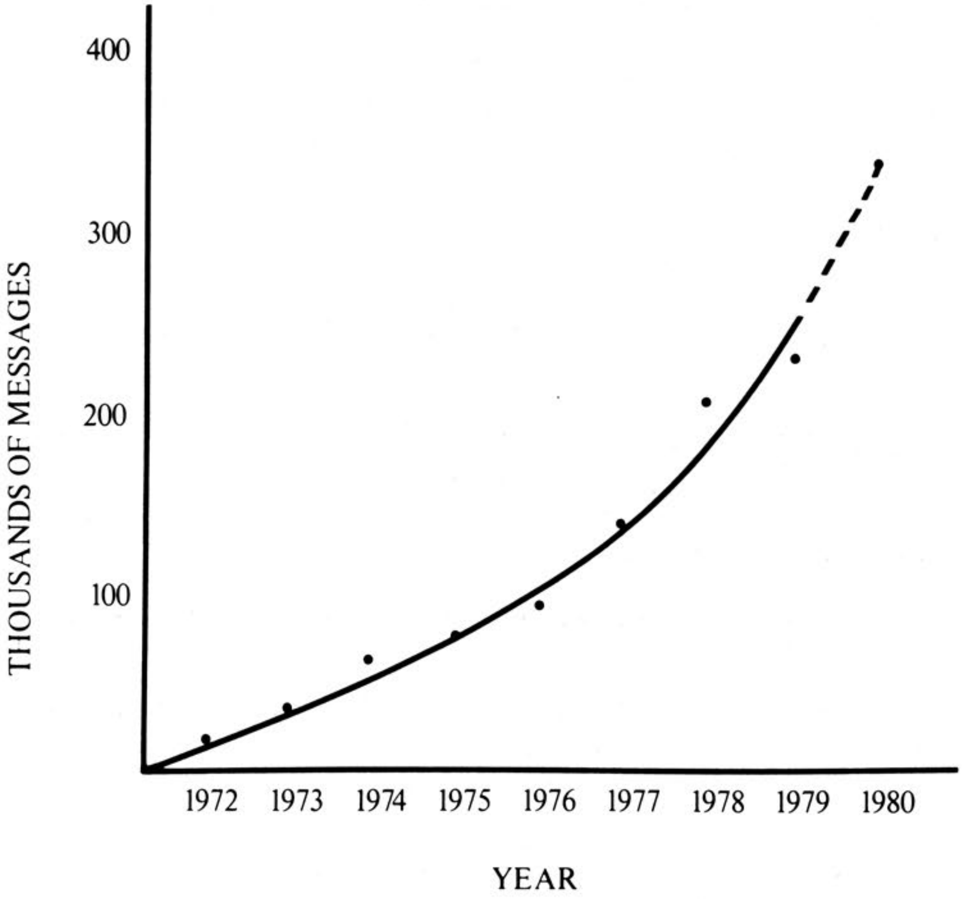
The Mailbox substitutes for many kinds of communication. When it is used in place of conventional mail services, delivery is instantaneous and mail awaits only the recipient's time to read it. When it is used in place of face-to-face dialogue or a telephone call, communication consumes far less time. People read about six times faster than they can talk, and thus are able to get the point across much more quickly with written communication. Mailbox messages also tend to be more succinct, since there is usually less non-task oriented information in them than a verbal conversation would graciously permit.

Managerial and professional personnel spend most of their time communicating with others, either verbally or in writing. As a result, even small impacts in their modes of communication can have a large economic benefit. While the use of the Mailbox is by no means advocated as a replacement for all other forms of communication (some tasks certainly require a verbal engagement), it has been found that the Mailbox has a profound positive impact on corporate communication patterns, often opening channels which did not previously exist.

Appendix: Statistical Information About the Mailbox

The Mailbox itself consists of two workspaces, and a supporting file which contains messages and user enrollment information (see the section entitled "Technical Aspects of the Mailbox" for more details). Because of the manner in which the workspaces and files are coupled, any number of independent Mailbox systems may be extant at a given time. Each Mailbox has its own set of workspaces and file, so that each may run different software if it desired to do so based on the community or organization it is serving.

There are presently about ten such Mailbox systems active, on the I.P. Sharp computers as well as on the computers of in-house customers. The first and largest of these is the I.P. Sharp Associates company and customer Mailbox, called `666 BOX`. This appendix contains statistical usage information for this Mailbox **only**; the combined usage levels including the other Mailbox systems running similar or identical software would be much higher. Present usage levels quoted herein are based on figures collected by the Mailbox during the first two quarters of 1980.



Annual Growth of Message Traffic in the Primary Mailbox

Figure 5

Figure 5 illustrates the growth of message traffic in the primary Mailbox for years 1972 to present. The projected volume of mail in 1980 is some 338,000 messages in this mailbox alone, and nearly a half million messages in 1981. More than 1,000 messages are now sent on a typical working day.

Over the years, the number of members in the Mailbox has grown from twenty-five people to almost 2,000, more than half of whom are customers. About forty percent of those people have mail pending at any given instant. The average incoming message level for users with a nonzero number pending is about seven messages.

The instantaneous capacity of the Mailbox is presently about 3,500 messages. (This number grows automatically according to usage patterns of the system.) For 3,500 messages, there are typically over 36,000 recipients, for a mean of around ten recipients per message. Roughly fifty-eight percent of all messages are addressed to one recipient. Of the forty-two percent that are not, the average number of recipients per message is approximately twenty. The maximum number of recipients actually addressed during the sample period was around 220, although a wider distribution list would have been possible just as easily.

On the order of nine percent of all mail sent through the Mailbox goes to a single group (which may contain any number of members). About 8.5 percent of the mail goes to a single group **and** one explicitly addressed person, who may or may not be a member of the group. (This situation arises most frequently when a message is addressed to an individual, and carbon-copied to a group.) The Mailbox presently contains some 400 groups, with an average enrollment of more than ten persons per group.

References

[Bair74] Bair, James H., "Evaluation and Analysis of an Augmented Knowledge Workshop", **Final Report for Phase I**, Rome Air Development Center, RADC-TR-74-79, April 1974.

[Barn78] Barna, Becky, "Electronic Messaging Can Make Cents", **Computer Decisions**, pp. 34-42, September 1978.

[Blea80] Bleackley, Beverley J., Special Article on Electronic Mail, **Computer Data**, pp. 30-31, February 1980.

[Druc74] Drucker, Peter F., **Management: Tasks, Responsibilities, Practices**. Harper and Row, New York, N.Y., 1974.

[Gold78] Goldsmith, Leslie H., **On the Security and Integrity of Programmed Systems**, Computer Systems Practice Unpublished Working Paper, Department of Electrical Engineering, University of Toronto, Toronto, Canada, November 1978.

[Gold80] Goldsmith, Leslie H., **The SHARP APL Messaging Processing Facility**. I.P. Sharp Associates Limited, Toronto, Canada, May 1980.

[Gott77] Gottheimer, Debra, "Mail the Postman Doesn't Carry". **Administrative Management**, pp. 37-50, March 1977.

[Grov79] Grove; George, "Information Management in the Office of the Future". **Management Review**, pp. 47-50, December 1979.

[Hark78] Harkness, Richard C., "Office Information Systems: An Overview and Agenda for Public Policy Research", **Telecommunications Policy**, June 1978.

[Konk76] Konkel, Gilbert J. and P.J. Peck, "Traditional Secretarial Cost Compared to Word Processing", **The Office**, pp. 67-68, February 1976.

[Mart73] Martin, James, **Design of Man-Computer Dialogues**, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1973.

[Mier80] Mier, Edwin E., "Tying Together Telephones and Typewriters", **Data Communications**, pp. 51-67, April 1980.

[Mint73] Mintzberg, Henry, **The Nature of Managerial Work**, Harper and Row, New York, N.Y., 1973.

[OEM80] **Office Equipment and Methods**, Special Article on Communicating Word Processors, p. 19, January 1980.

[Pank76] Panko, Raymond R., Presentation to the Seminar on the Augmented

Knowledge Workshop, Stanford Research Institute, Menlo Park, Ca., November 1976.

[Pott77] Potter, David A., "Software Objectives for the Administrative Network", International Data Corporation Executive Conference, Ft. Lauderdale, Fla., November 1977.

[Rowa78] Rowan, Roy, "Keeping the Clock from Running Out", **Fortune Magazine**, p. 76, November 1978.

[Uhli79] Uhlig, Ronald P., D.J. Farber, and J.H. Bair, **The Office of the Future**, vol. 1, International Council for Computer Communications, North-Holland Publishing Company, Amsterdam, The Netherlands, August 1979.

APL - THE SCHEDULER'S TOOL

Daneda K. Cullen
Manager, Crew Scheduling
Air California
Newport Beach, California

The Airline

Air California, intra-state carrier, was born in 1966. Service began between Orange County and San Francisco airports, with two Lockheed Electras. Two DC-9 Jets were added as service expanded. Six Boeing 737 Jets were introduced in 1968.

At present, our fleet consists of fourteen B-737's with two DC-9-80's to be delivered in October of this year. The carrier is run inter-state, serving thirteen cities.

The Scheduling Department

May I introduce you to Crew Scheduling. Perhaps with a brief explanation of the terminology used at Air California and the functions of the scheduler, you might better understand this report and our need for APL.

Glossary

Master Bid Times (Figure 1): Average time, block-to-block between points served on a scheduled basis, determined by averaging actual times flown.

Trip Sequence: Series of trip segments, starting with initial departure from a domicile and ending with final arrival at domicile, which are combined for preparation of the crew members' schedule.

Pay Credits: Time for which a crew member is paid while not engaged in the actual flying of an aircraft —

a) Deadhead times: time spent in traveling from one point to another either for duty or returning from duty.
b) Duty/trip rig: pay adjustment prorated for the total elapsed time away from domicile.

Duty Aloft: Flight time during which a crew member is engaged in the operation of an aircraft.

Flow Chart (Figure 2): Aircraft schedule.

Finder (Figure 3): A listing of trip sequences, derived from the flow chart and used to assign work schedules for crew members.

Bidlines (Figure 4): Monthly work schedules comprised of trip sequences and days off. The lines of work are bid for by individual crew members. The line values determine the crew members' minimum pay for the month. The Bidlines are the end product of work by the schedulers. During the month, the scheduler makes adjustments to scheduled work, attempting to maintain the projected number of hours to be worked (Crew Tracking).

In the early years of Air California, one scheduler manager planned the work for 120 flight and cabin crew members. That department has evolved into one with seven people managing over four hundred flight personnel.

Job Description

Responsible for advance planning and efficient, economical utilization of flight crews, through application of Company policies, F.A.A. regulations and union contract agreements.

Principal Duties

- Responsible for building finders from finished flow charts.
- Construct monthly bidlines.
- Schedule ground school and proficiency training for pilots.
- Reschedule crew members to accomodate changes resulting from holidays, charter operation, weather and mechanical problems.
- Trip sequence coverage due to vacation, illness or crew illegalities.
- Calculate accumulated flight time and project remaining time with adjustments to ensure continued contract and F.A.A. requirements.
- Monitor crew check-ins to ensure trip coverage and on-time performance.
- Coordinate and arrange flight crew ground transportation and hotel accomodations.
- Compile a daily work sheet with trip sequences, check-in times and crew names.
- Maintain daily, weekly and monthly records including all payroll-related information as required for Accounting and Flight Operations.
- Coordinate and integrate the meshing of schedules from one bid period to the next, and between schedule changes that occur during a bid period.

This complex set of duties coupled with increased scheduling responsibilities, demands for more detailed statistical reports, maintenance of records, or simply the solution to "Growth", has prompted computerization.

APL as a Tool

Our search for automation began in 1977. What we were looking for was a system that would assist in preparing the monthly reports and schedules called for by the flight crew work contracts. We were faced with two disappointing years. Various companies would accept, then decline the challenge of what seemed to be a complicated scheduling system. Most software development requires extensive specification of the final results of the system. Since Air California had little or no experience with data processing and what was difficult or easy to accomplish, most vendors were discouraging about their ability to produce an acceptable system.

The various scheduling systems used by other carriers were investigated. Most such systems require in-house computer facilities not available at Air California. Also, major changes to Air California's operating procedures would have been required to adapt to another carrier's system.

We were introduced to I.P. Sharp through our Marketing Department. In January of 1978, Sharp automated the marketing statistical functions and seemed very interested in developing a crew scheduling system. After initial discussions of the problems of crew scheduling, work on a system began by an IPSA person observing the work of the scheduling department for a week. Out of the week of observation came some recommendations on what was needed and feasible to implement. The scheduling problem is part of the total operations aspect of the airline. The idea of a system which would tie together the functions of scheduling, operations, and market planning emerged from the study.

The SHARP APL system is ideally suited to implementing such an integrated system. APL allows the modular development needed to work independently on different aspects of the problem while a convenient, powerful file system allows easy interchange of data between obvious parts of the system.

May I comment before proceeding, that this report will not yield technical computer information. I am not an APL expert. I have established a need for a computer system and can only analyze the results in terms of their impact on the crew scheduling department.

The system being developed has two parts —

I. Crew Scheduling
II. Crew Tracking and Operations Reporting

The goal of I.P. Sharp was to create interactive systems linking Crew Scheduling, Dispatch/Operations Control and Marketing. The data collected would generate management information for better route analysis, schedule planning and crew management, and thus enable Air California to operate more efficiently and more profitably. Figure 5 shows how data may be shared between various users of the crew scheduling and tracking system.

I. Crew Scheduling

APL's function: Optimum utilization of crews. Our utilization goal is to generate trip sequences with a minimum amount of non-productive time or "pay credits". The difficulty of this task is increased by a unique problem of Air California's: the domicile is Orange County which has an airport restriction on take-off and landings per carrier. Our maximum is twenty-eight per day. This restriction leads to aircraft rotation through Ontario, seventy miles from Orange County. Duty time limitations on crews force a trip replacement often in Ontario. The cost is astronomical. We must provide ground transportation in addition to crew pay and credit for the deadhead to and from domicile. Pay and credit is synonymous with crew compliment as all pay (master bid plus pay credits) is credited toward total maximum flight hours which determine the number of flight crews needed per bid period.

The critical inceptive point is finder building. All flights per aircraft rotation are paired based on contract and F.A.A. regulations to form trip sequences. The manual system generated approximately a pay credit average of ten percent of total hours. A finder was produced in twenty-four man-hours.

The machine solution has created a more efficient finder, averaging seven percent pay credit hours, a reduction of one hundred hours per month. The summary sheet of pay credits (Figure 6) gives a breakdown of sequence values per day, pay credit hours with percentage equivalent per day. This report provides information on aircraft rotations for any given day of the week which are creating excessive non-productive time in the scheduling of crews. A marketing analysis can then be made on the profitability of rerouting an aircraft to reduce pay credit cost.

Finder production is accomplished in less than one minute. Listed are additional reports derived from a completed finder —

Taxi Report (Figure 7): Ground transportation requirement per day with reference to applicable sequence, departure time from domicile and arrival of inbound flight.

Hotel Report (Figure 8): Room requirement per day listing applicable sequence, overnight station with arrival time.

Both reports assist the scheduler in arranging for hotel accomodations and transportation in an efficient and expeditious manner.

Bidlines are derived from the completed finder. A Flight Time and Pay Projection Sheet is generated (Figure 9) to analyze hours to be flown and total trip sequences to cover. A contract manning formula is applied to total hours to arrive at total number of bidlines. The resulting figure is examined to determine available working days for total sequence coverage. The computer then generates a work schedule combining legal work rules governed by labor contract and F.A.A. regulations.

Examples of rules:

- required number of days free from duty
- duty aloft limitations in twenty-four hour timeframe
- duty aloft during any consecutive seven day period
- domicile rest period between trip sequences
- maximum flight pay per bidline

The manual system was completed in forty man-hours. The computer solution takes but one hour.

II. Crew Tracking

Currently under development.

System Overview

To allow award of bidlines to crews and allow alterations to these assignments. To keep track of open-time (Schedule Mesh, Trip Trades, Vacation, Training, Sick Hours).

To monitor crew check-in. To record actual flight and crew activity and run crew projections. To monitor crew legalities and advise of developing problems. To assist the scheduler in choosing open-time candidates. To simulate the impact of various rescheduling solutions. To allow the reassignments of trip segments.

To allow maintenance of employee records of qualifications. To indicate qualification renewals, to generate pay (Sick, Training, Vacation, Status, Banked Hours).

To allow analysis of monthly crew utilization, crew activity, reasons for delays. To allow analysis of scheduling changes in relation to flight or bidline characteristics.

Activities

A. Crew Detail Data
   Stored per crew member, updated as appropriate
   1. Dates of hire, furlough, leaves of absence
   2. Contact information (name, address)
   3. Awarded vacation time

B. Crew Reference Data
   Stored per crew member, updated monthly as appropriate
   1. Category, status
   2. Type of bidholder
   3. Qualifications (equipment rating)
   4. Pay rate

C. Crew Scheduled and Actual Activity
   Stored per crew member, scheduled or projected and actual per month (original source of this data: awarded lines)
   1. Day
   2. Type of activity or reason for pay (flying, vacation, raise to guarantee, training, sick)
   3. Hours
   4. Pay value
   5. Check in/out times
   6. Source of assignment (scheduled, rescheduled)

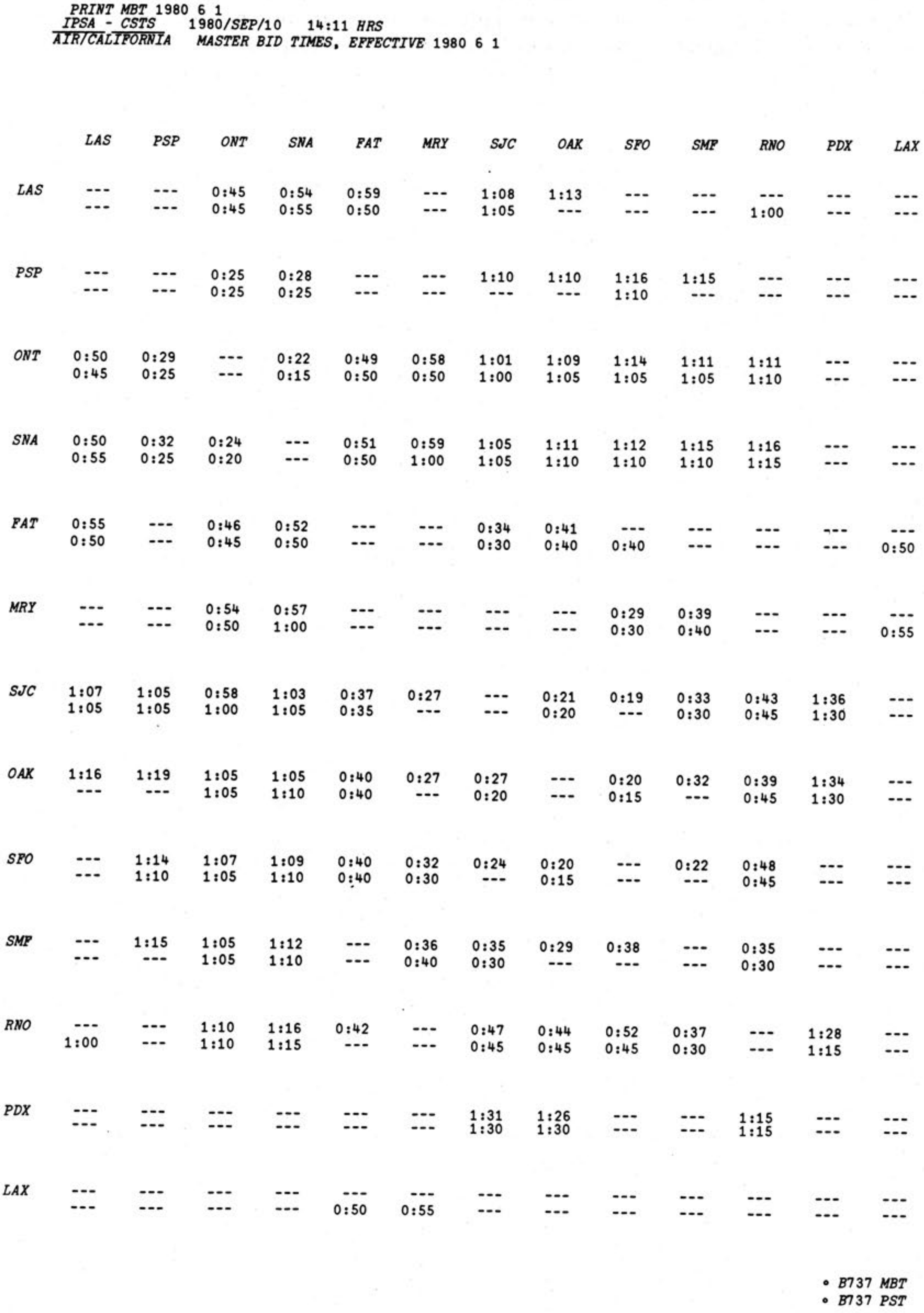
D. Trip Trading History
   Stored per crew member, per current month

Reports

A. Crew member Pay Report
B. Crew member Expense Report (per diem, motels, transportation)
C. Monthly Crew Utilization Report
D. Quarterly Reserve, Crew member Average Flight Pay Reports
E. Banked Hours (accrual, withdrawal)

Conclusion

APL is indeed our tool towards better management. We are in the beginning phase of on-line utilization and have much more to learn about this decision support system. Its flexibility has greatly influenced our daily operation and our department forecasts, and will play a tremendous role in our future labor negotiations.



```
PRINT MBT 1980 6 1
IPSA - CSTS     1980/SEP/10    14:11 HRS
AIR/CALIFORNIA   MASTER BID TIMES, EFFECTIVE 1980 6 1
```

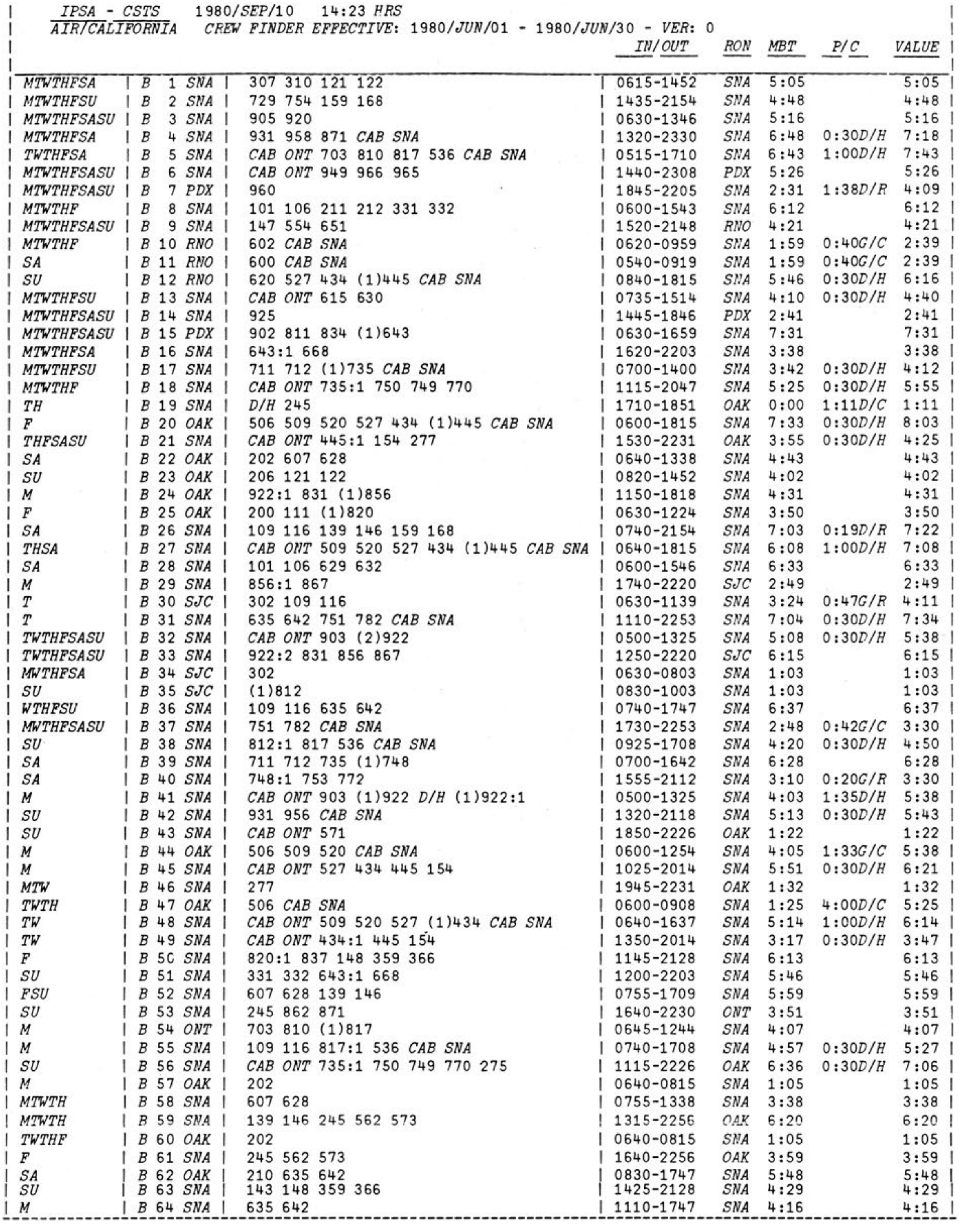
	LAS	PSP	ONT	SNA	FAT	MRY	SJC	OAK	SFO	SMF	RNO	PDX	LAX
LAS	--- ---	--- ---	0:45 0:45	0:54 0:55	0:59 0:50	--- ---	1:08 1:05	1:13 ---	--- ---	--- ---	--- 1:00	--- ---	--- ---
PSP	--- ---	--- ---	0:25 0:25	0:28 0:25	--- ---	--- ---	1:10 ---	1:10 ---	1:16 1:10	1:15 ---	--- ---	--- ---	--- ---
ONT	0:50 0:45	0:29 0:25	--- ---	0:22 0:15	0:49 0:50	0:58 0:50	1:01 1:00	1:09 1:05	1:14 1:05	1:11 1:05	1:11 1:10	--- ---	--- ---
SNA	0:50 0:55	0:32 0:25	0:24 0:20	--- ---	0:51 0:50	0:59 1:00	1:05 1:05	1:11 1:10	1:12 1:10	1:15 1:10	1:16 1:15	--- ---	--- ---
FAT	0:55 0:50	--- ---	0:46 0:45	0:52 0:50	--- ---	--- ---	0:34 0:30	0:41 0:40	--- 0:40	--- ---	--- ---	--- ---	--- 0:50
MRY	--- ---	--- ---	0:54 0:50	0:57 1:00	--- ---	--- ---	--- ---	--- ---	0:29 0:30	0:39 0:40	--- ---	--- ---	--- 0:55
SJC	1:07 1:05	1:05 1:05	0:58 1:00	1:03 1:05	0:37 0:35	0:27 ---	--- ---	0:21 0:20	0:19 ---	0:33 0:30	0:43 0:45	1:36 1:30	--- ---
OAK	1:16 ---	1:19 ---	1:05 1:05	1:05 1:10	0:40 0:40	0:27 ---	0:27 0:20	--- ---	0:20 0:15	0:32 ---	0:39 0:45	1:34 1:30	--- ---
SFO	--- ---	1:14 1:10	1:07 1:05	1:09 1:10	0:40 0:40	0:32 0:30	0:24 ---	0:20 0:15	--- ---	0:22 ---	0:48 0:45	--- ---	--- ---
SMF	--- ---	1:15 ---	1:05 1:05	1:12 1:10	--- ---	0:36 0:40	0:35 0:30	0:29 ---	0:38 ---	--- ---	0:35 0:30	--- ---	--- ---
RNO	--- 1:00	--- ---	1:10 1:10	1:16 1:15	0:42 ---	--- ---	0:47 0:45	0:44 0:45	0:52 0:45	0:37 0:30	--- ---	1:28 1:15	--- ---
PDX	--- ---	--- ---	--- ---	--- ---	--- ---	--- ---	1:31 1:30	1:26 1:30	--- ---	--- ---	1:15 1:15	--- ---	--- ---
LAX	--- ---	--- ---	--- ---	--- ---	--- 0:50	--- 0:55	--- ---	--- ---	--- ---	--- ---	--- ---	--- ---	--- ---

∘ B737 MBT
∘ B737 PST

Figure 1

```
IPSA - CSTS    1980/SEP/10    14:15 HRS  •  AIR/CALIFORNIA MBT FLOWCHART
FOR - F     EFFECTIVE  1980/APR/01 - 1980/APR/30 [VER 1]

B737 - 1   B737 - 2   B737 - 3   B737 - 4   B737 - 5   B737 - 6   B737 - 7   B737 - 8   B737 - 9   B737 -10   B737 -11   B737 -12
                                                                              [504]
            [302]                  [101]                  [202]                 OAK 0630    [903]      [602]      [902]      [200]
 [307]     SJC 0700    [905]     SNA 0700    [703]     OAK 0710              FAT 0710   ONT 0700   RNO 0650   PDX 0700   OAK 0700
SNA 0715   SNA 0803   SNA 0730   SFO 0812   ONT 0715   SNA 0815    [711]     FAT 0725   SJC 0801   SFO 0742   SJC 0835   SFO 0726
SJC 0820              OAK 0841              SMF 0830              SNA 0800   ONT 0811   SJC 0820   SFO 0752   SJC 0845   SFO 0745
                      OAK 0900                                    SMF 0915              PDX 0955   ONT 0859   ONT 0943   SNA 0854
            [109]     PDX 1030    [106]      [810]      [607]                                                 ONT 1000
 [310]     SNA 0840              SFO 0840   SMF 0855   SNA 0855               [509]                           LAS 1050
SJC 0905   SFO 0952              SNA 0949   SNA 1007   SJC 1000    [712]     ONT 0840
SNA 1008                                    SNA 1025   SJC 1015   SMF 0940   SFO 0950
                                            LAS 1115   RNO 1058   SJC 1015                          [615]                 [111]
                                                                  SJC 1025               [922]     ONT 0935              SNA 0930
 [121]      [116]                  [211]                          ONT 1123    [520]     PDX 1030   FAT 1024              SFO 1042
SNA 1045   SFO 1030    [920]     SNA 1020                         ONT 1140   SFO 1015   OAK 1200   FAT 1045
MRY 1144   SNA 1139   PDX 1100   FAT 1111                         PSP 1209   MRY 1043   OAK 1220   OAK 1126    [811]
MRY 1200              RNO 1215               [817]      [628]                MRY 1100   SNA 1325   OAK 1145   LAS 1110    [820]
SFO 1229              RNO 1230    [212]     LAS 1150   RNO 1130              ONT 1154   SNA 1350   RNO 1224   ONT 1155   SFO 1115
            [635]     SNA 1346   FAT 1140   SNA 1240   SJC 1217                         LAS 1440              ONT 1215   SNA 1224
           SNA 1210              SNA 1232   SNA 1300   SJC 1235                                               SJC 1316   SNA 1245
           SFO 1322                         OAK 1411   SNA 1338    [735]      [527]                                      LAS 1335
           SFO 1340                                               PSP 1235   ONT 1225
 [122]     RNO 1428               [331]                           ONT 1300   MRY 1323
SFO 1310                         SNA 1300                         ONT 1315   MRY 1333               [630]      [834]
MRY 1338               [931]     SJC 1405               [139]     SJC 1416   SFO 1402              RNO 1300   SJC 1350
MRY 1355              SNA 1420               [536]     SNA 1415   SJC 1430                         SFO 1352   FAT 1427
SNA 1452    [642]     RNO 1536              OAK 1435   SFO 1527   SMF 1503    [434]      [831]     SFO 1405   FAT 1440    [837]
           RNO 1455   RNO 1550    [332]     SJC 1502                         SFO 1430   LAS 1510   SNA 1514   LAS 1535   LAS 1410
           OAK 1539   PDX 1705   SJC 1440   SJC 1512                         ONT 1537   FAT 1609                         SNA 1500
           OAK 1600              SNA 1543   ONT 1610    [146]                ONT 1550   FAT 1619                         SNA 1525
 [729]     FAT 1640                                    SFO 1600              PSP 1619   SJC 1653                         SFO 1637
SNA 1535   FAT 1655                                    SNA 1709    [750]                                       [643]
SMF 1650   SNA 1747               [147]                           SMF 1545                          [925]     LAS 1605
                       [956]     SNA 1620    [949]                ONT 1650    [445]                SNA 1545   SNA 1655
                      PDX 1735   SFO 1732   ONT 1640    [245]                PSP 1650    [856]     SJC 1650   SNA 1720
 [754]                SJC 1910              OAK 1749   SNA 1740              ONT 1715   SJC 1715   SJC 1710   SJC 1825    [148]
SMF 1720              SJC 1920              OAK 1800   OAK 1851    [749]     ONT 1730   SNA 1818   PDX 1845   SJC 1845   SFO 1710
SNA 1832              ONT 2018              PDX 1930              ONT 1720   SFO 1840   SNA 1840              RNO 1928   SNA 1819
            [751]                  [554]                          SJC 1821              LAS 1930
           SNA 1830              SFO 1800               [862]     SJC 1835
           SMF 1945              ONT 1907              OAK 1915   SMF 1908    [154]                                       [359]
 [159]                                                 ONT 2020              SFO 1905                                    SNA 1850
SNA 1900                          [651]      [966]     ONT 2035              SNA 2014               [960]                SJC 1955
SFO 2012    [782]      [571]     ONT 1935   PDX 2000   LAS 2125                          [867]     PDX 1915    [668]
           SMF 2010   ONT 2050   SFO 2045   RNO 2115               [770]                LAS 2005   OAK 2045   RNO 2000
           SJC 2045   SJC 2151   SFO 2100                         SMF 1935    [277]     SNA 2055   OAK 2100   SJC 2047    [366]
 [168]     SJC 2055   SJC 2205   RNO 2148                         SNA 2047   SNA 2045   SNA 2115   SNA 2205   SJC 2100   SJC 2025
SFO 2045   ONT 2153   OAK 2226               [965]      [871]                FAT 2136   SJC 2220              SNA 2203   SNA 2128
SNA 2154                                    RNO 2140   LAS 2145              FAT 2150
                                            PDX 2255   ONT 2230              OAK 2231
```

Figure 2

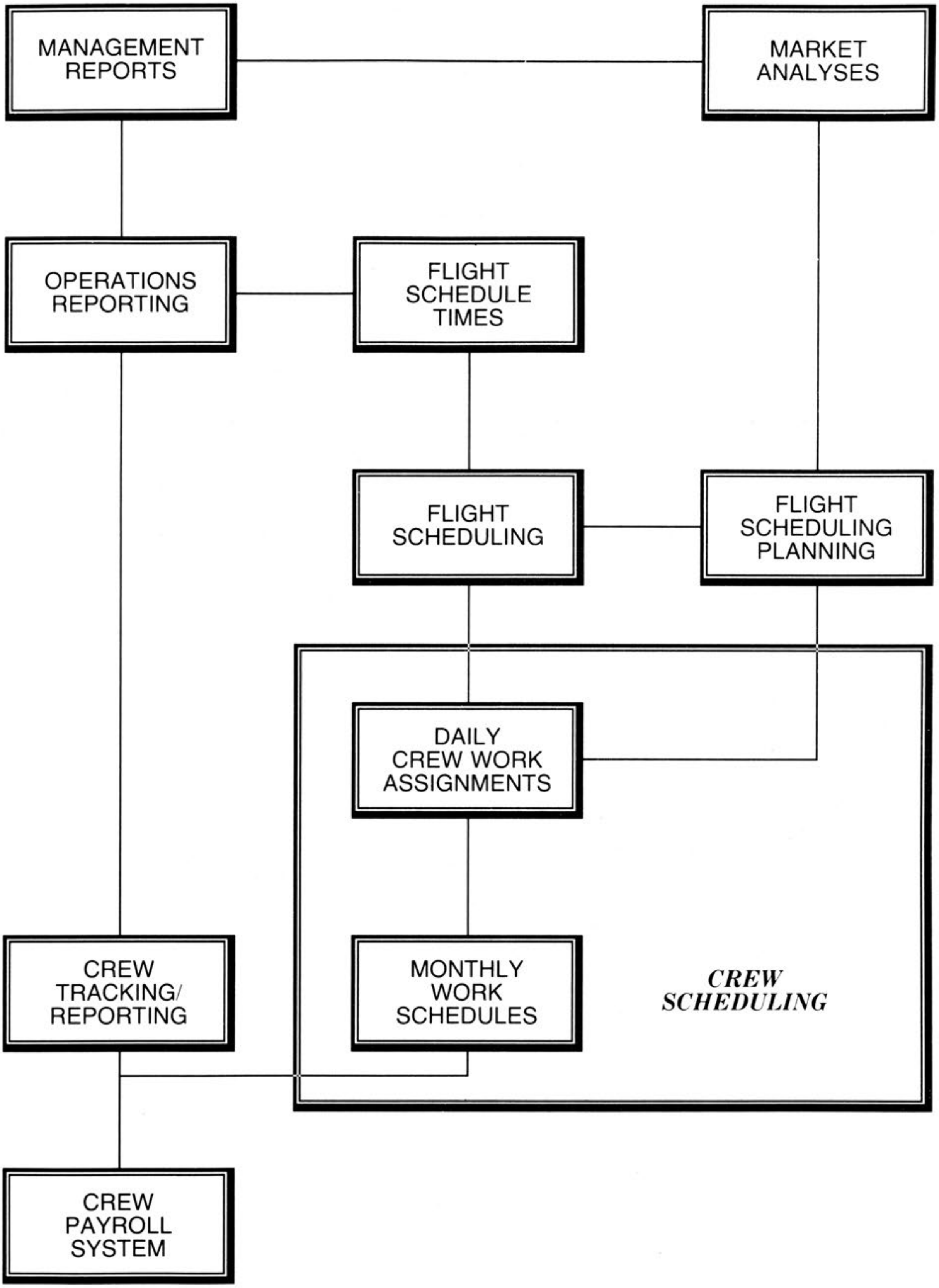


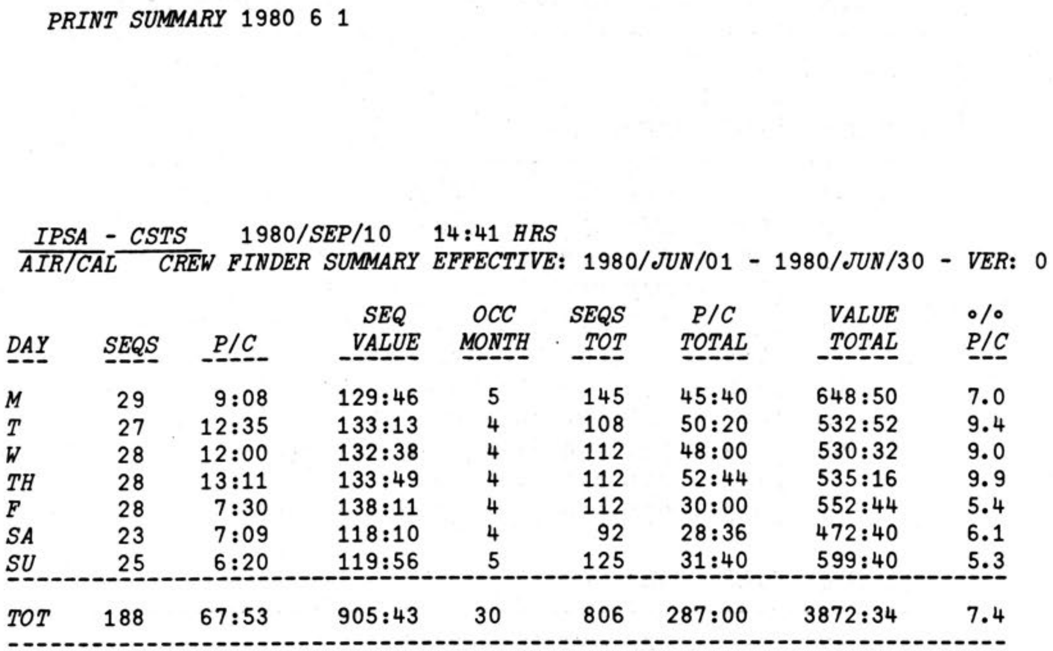
```
|   IPSA - CSTS    1980/SEP/10   14:23 HRS                                                                  |
|   AIR/CALIFORNIA   CREW FINDER EFFECTIVE: 1980/JUN/01 - 1980/JUN/30 - VER: 0                              |
|                                                              IN/OUT     RON  MBT   P/C      VALUE |
|                                                                                                           |
| MTWTHFSA   | B  1 SNA | 307 310 121 122                        | 0615-1452  SNA  5:05            5:05 |
| MTWTHFSU   | B  2 SNA | 729 754 159 168                        | 1435-2154  SNA  4:48            4:48 |
| MTWTHFSASU | B  3 SNA | 905 920                                | 0630-1346  SNA  5:16            5:16 |
| MTWTHFSA   | B  4 SNA | 931 958 871 CAB SNA                    | 1320-2330  SNA  6:48  0:30D/H   7:18 |
| TWTHFSA    | B  5 SNA | CAB ONT 703 810 817 536 CAB SNA        | 0515-1710  SNA  6:43  1:00D/H   7:43 |
| MTWTHFSASU | B  6 SNA | CAB ONT 949 966 965                    | 1440-2308  PDX  5:26            5:26 |
| MTWTHFSASU | B  7 PDX | 960                                    | 1845-2205  SNA  2:31  1:38D/R   4:09 |
| MTWTHF     | B  8 SNA | 101 106 211 212 331 332                | 0600-1543  SNA  6:12            6:12 |
| MTWTHFSASU | B  9 SNA | 147 554 651                            | 1520-2148  RNO  4:21            4:21 |
| MTWTHF     | B 10 RNO | 602 CAB SNA                            | 0620-0959  SNA  1:59  0:40G/C   2:39 |
| SA         | B 11 RNO | 600 CAB SNA                            | 0540-0919  SNA  1:59  0:40G/C   2:39 |
| SU         | B 12 RNO | 620 527 434 (1)445 CAB SNA             | 0840-1815  SNA  5:46  0:30D/H   6:16 |
| MTWTHFSU   | B 13 SNA | CAB ONT 615 630                        | 0735-1514  SNA  4:10  0:30D/H   4:40 |
| MTWTHFSASU | B 14 SNA | 925                                    | 1445-1846  PDX  2:41            2:41 |
| MTWTHFSASU | B 15 PDX | 902 811 834 (1)643                     | 0630-1659  SNA  7:31            7:31 |
| MTWTHFSA   | B 16 SNA | 643:1 668                              | 1620-2203  SNA  3:38            3:38 |
| MTWTHFSU   | B 17 SNA | 711 712 (1)735 CAB SNA                 | 0700-1400  SNA  3:42  0:30D/H   4:12 |
| MTWTHF     | B 18 SNA | CAB ONT 735:1 750 749 770              | 1115-2047  SNA  5:25  0:30D/H   5:55 |
| TH         | B 19 SNA | D/H 245                                | 1710-1851  OAK  0:00  1:11D/C   1:11 |
| F          | B 20 OAK | 506 509 520 527 434 (1)445 CAB SNA     | 0600-1815  SNA  7:33  0:30D/H   8:03 |
| THFSASU    | B 21 SNA | CAB ONT 445:1 154 277                  | 1530-2231  OAK  3:55  0:30D/H   4:25 |
| SA         | B 22 OAK | 202 607 628                            | 0640-1338  SNA  4:43            4:43 |
| SU         | B 23 OAK | 206 121 122                            | 0820-1452  SNA  4:02            4:02 |
| M          | B 24 OAK | 922:1 831 (1)856                       | 1150-1818  SNA  4:31            4:31 |
| F          | B 25 OAK | 200 111 (1)820                         | 0630-1224  SNA  3:50            3:50 |
| SA         | B 26 SNA | 109 116 139 146 159 168                | 0740-2154  SNA  7:03  0:19D/R   7:22 |
| THSA       | B 27 SNA | CAB ONT 509 520 527 434 (1)445 CAB SNA | 0640-1815  SNA  6:08  1:00D/H   7:08 |
| SA         | B 28 SNA | 101 106 629 632                        | 0600-1546  SNA  6:33            6:33 |
| M          | B 29 SNA | 856:1 867                              | 1740-2220  SJC  2:49            2:49 |
| T          | B 30 SJC | 302 109 116                            | 0630-1139  SNA  3:24  0:47G/R   4:11 |
| T          | B 31 SNA | 635 642 751 782 CAB SNA                | 1110-2253  SNA  7:04  0:30D/H   7:34 |
| TWTHFSASU  | B 32 SNA | CAB ONT 903 (2)922                     | 0500-1325  SNA  5:08  0:30D/H   5:38 |
| TWTHFSASU  | B 33 SNA | 922:2 831 856 867                      | 1250-2220  SJC  6:15            6:15 |
| MWTHFSA    | B 34 SJC | 302                                    | 0630-0803  SNA  1:03            1:03 |
| SU         | B 35 SJC | (1)812                                 | 0830-1003  SNA  1:03            1:03 |
| WTHFSU     | B 36 SNA | 109 116 635 642                        | 0740-1747  SNA  6:37            6:37 |
| MWTHFSASU  | B 37 SNA | 751 782 CAB SNA                        | 1730-2253  SNA  2:48  0:42G/C   3:30 |
| SU         | B 38 SNA | 812:1 817 536 CAB SNA                  | 0925-1708  SNA  4:20  0:30D/H   4:50 |
| SA         | B 39 SNA | 711 712 735 (1)748                     | 0700-1642  SNA  6:28            6:28 |
| SA         | B 40 SNA | 748:1 753 772                          | 1555-2112  SNA  3:10  0:20G/R   3:30 |
| M          | B 41 SNA | CAB ONT 903 (1)922 D/H (1)922:1        | 0500-1325  SNA  4:03  1:35D/H   5:38 |
| SU         | B 42 SNA | 931 956 CAB SNA                        | 1320-2118  SNA  5:13  0:30D/H   5:43 |
| SU         | B 43 SNA | CAB ONT 571                            | 1850-2226  OAK  1:22            1:22 |
| M          | B 44 OAK | 506 509 520 CAB SNA                    | 0600-1254  SNA  4:05  1:33G/C   5:38 |
| M          | B 45 SNA | CAB ONT 527 434 445 154                | 1025-2014  SNA  5:51  0:30D/H   6:21 |
| MTW        | B 46 SNA | 277                                    | 1945-2231  OAK  1:32            1:32 |
| TWTH       | B 47 OAK | 506 CAB SNA                            | 0600-0908  SNA  1:25  4:00D/C   5:25 |
| TW         | B 48 SNA | CAB ONT 509 520 527 (1)434 CAB SNA     | 0640-1637  SNA  5:14  1:00D/H   6:14 |
| TW         | B 49 SNA | CAB ONT 434:1 445 154                  | 1350-2014  SNA  3:17  0:30D/H   3:47 |
| F          | B 50 SNA | 820:1 837 148 359 366                  | 1145-2128  SNA  6:13            6:13 |
| SU         | B 51 SNA | 331 332 643:1 668                      | 1200-2203  SNA  5:46            5:46 |
| FSU        | B 52 SNA | 607 628 139 146                        | 0755-1709  SNA  5:59            5:59 |
| SU         | B 53 SNA | 245 862 871                            | 1640-2230  ONT  3:51            3:51 |
| M          | B 54 ONT | 703 810 (1)817                         | 0645-1244  SNA  4:07            4:07 |
| M          | B 55 SNA | 109 116 817:1 536 CAB SNA              | 0740-1708  SNA  4:57  0:30D/H   5:27 |
| SU         | B 56 SNA | CAB ONT 735:1 750 749 770 275          | 1115-2226  OAK  6:36  0:30D/H   7:06 |
| M          | B 57 OAK | 202                                    | 0640-0815  SNA  1:05            1:05 |
| MTWTH      | B 58 SNA | 607 628                                | 0755-1338  SNA  3:38            3:38 |
| MTWTH      | B 59 SNA | 139 146 245 562 573                    | 1315-2256  OAK  6:20            6:20 |
| TWTHF      | B 60 OAK | 202                                    | 0640-0815  SNA  1:05            1:05 |
| F          | B 61 SNA | 245 562 573                            | 1640-2256  OAK  3:59            3:59 |
| SA         | B 62 OAK | 210 635 642                            | 0830-1747  SNA  5:48            5:48 |
| SU         | B 63 SNA | 143 148 359 366                        | 1425-2128  SNA  4:29            4:29 |
| M          | B 64 SNA | 635 642                                | 1110-1747  SNA  4:16            4:16 |
---------------------------------------------------------------------------------------------------------
```

Figure 3

```
    IPSA - CSTS    1980/SEP/10    15:12 HRS
    AIR/CALIFORNIA CREW SCHEDULE EFFECTIVE   JUN 1980
------------------------------------------------------------

                        SU   M  TU   W  TH   F  SA  SU   M  TU   W  TH   F  SA  SU   M  TU   W  TH   F  SA  SU   M  TU   W  TH   F  SA  SU   M
|                |  |    1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | DC S F  TT |
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                | 1|    X  19 ^20   -  32 ^35   X   X   X  28 ^29  13   -   X   X   2  18 ^19   3   X   X   X  14  17  13   X   X  55 ^56   8 | 13 6 5 7920|
|                |  |    - 512 227   - 553 110   -   -   - 200 637 539   -   -   - 541 329 403 611   -   -   - 410 456 539   -   - 703 110 800 |    0 012633|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                | 2|   24 ^25   -   2   X   X   X  10   4  ^5   -   X   X  18 ^19   X   X  11 ^12   9  46   8   X   X   X  31  10  53 ^54   X | 13 3 5 7910|
|                |  |  614 103   - 541   -   -   - 509 251 735   -   -   - 329 403   -   - 157 503 503 550 800   -   -   - 455 509 334 334   - |    0 011545|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                | 3|   45   5  39   -   X   X  11 ^12  32 ^33   X   X  48 ^49  28 ^29   X   X   X  32 ^34  27   X   X   X  13  20  21   -   X | 13 1 5 7901|
|                |  |  616 408 654   -   -   - 157 503 553 135   -   - 537 123 200 637   -   -   - 553 135 703   -   -   - 539 520 608   -   - |    0 013111|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                | 4|   44  35  36   -   X   X  53 ^54  30   X   X  11 ^12  13   2   X   X   X  22 ^23   -   3   X   X   X  25 ^26  39   8   X | 13 2 4 7937|
|                |  |  330 330 611   -   -   - 334 334 542   -   - 157 503 539 541   -   -   - 328 332   - 611   -   -   - 527 245 553 800   - |    0 011937|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                | 5|    X  36   -   4  ^5   X   X   6  ^7  24   -  21   8   X   X   X  28 ^29   -   X   X   X   8  28 ^29   -   X   X  24  14 | 15 7 4 7902|
|                |  |    - 611   - 251 735   -   - 312 401 550   - 608 800   -   -   - 200 637   -   -   -   - 800 200 637   -   -   - 550 410 |    0 012124|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                | 6|   10  22  10   X   X   X  52  17   -   1   X   X  25 ^26   1   X   X   X  31  48 ^49   2   X   X  32 ^33   9  13   -   X | 13 1 3 7904|
|                |  |  526 338 526   -   -   - 458 456   - 521   -   - 527 245 521   -   -   - 455 537 123 541   -   - 553 135 503 539   -   - |    0 012206|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                | 7|   13 ^14  29   -  17   X   X   X   X  20  10  37 ^38   X   X  28 ^29  10   -   X   X   X  30   4  ^5   -   X   X   4  ^5 | 14 7 5 7907|
|                |  |  225 518 648   - 456   -   -   -   - 520 509 426 425   -   - 200 637 509   -   -   -   - 542 251 735   -   -   - 251 735 |    0 012004|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                | 8|   15 ^16  12   X   X  32 ^34  32 ^33   X   X   X  11 ^12   8   -   X   X   X  20   6  ^7   -   X   X  21   4  ^5   2   X | 13 0 6 7901|
|                |  |  435 442 500   -   - 553 135 553 135   -   -   - 157 503 800   -   -   -   - 520 312 401   -   -   - 608 251 735 541   - |    0 012252|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                | 9|    7  ^8  27 ^16   -   X   X  15 ^16   8   X   X   X   6  ^7  27  15 ^16   X   X   X   X  15 ^16  27   2   -   X   X  25 | 13 7 7 7913|
|                |  |  410 250 342 334   -   -   - 436 334 800   -   -   - 312 401 703 436 334   -   -   -   - 436 334 703 541   -   -   - 527 |    0 012741|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                |10|    X   X   X  24  37 ^38  21   X   X  31  31  41   -   X   X   3   -  27   X   1   9   X   X   X  17   -   2  40   -   9 | 15 2 1 7902|
|                |  |    -   -   - 550 426 425 608   -   - 455 455 505   -   -   - 611   - 703   - 521 503   -   -   - 456   - 541 400   - 503 |    0 012255|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                |11|    X   X   X  32 ^33  20  30   X   X   X   9  22 ^23  50   X   X  11 ^12  41  43   -   9   X   X   X  42  48 ^49   1   - | 13 1 4 7908|
|                |  |    -   -   - 553 135 520 542   -   -   - 503 328 332 759   -   - 157 503 505 553   - 503   -   -   - 514 537 123 521   - |    0 012235|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                |12|    -  27 ^28   X   X   X  45  24  14   X   X   X   1  31   9   X   X  13  13   -  31   1   X   X   X  17   -  46  31   2 | 14 3 1 7901|
|                |  |    - 342 334   -   -   - 330 550 410   -   -   - 521 455 503   -   - 539 539   - 455 521   -   -   - 456   - 550 455 541 |    0 012144|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                |13|    X   X   X  17  42   4  ^5   X   X  10   2  40   -   X   X  31  14  14  40   X   X   X  32 ^33  20   X  28 ^29   -  31 | 13 4 3 7901|
|                |  |    -   -   - 456 514 251 735   -   - 509 541 400   -   -   - 455 410 410 400   -   -   - 553 135 520   - 200 637   - 455 |    0 012113|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                |14|    X  21  15  ^7  25 ^26   X   X   X  17   -   2  30   X   X   1   6  ^7  39   -   X   X   1   1  10   X   X   X   6  ^7 | 13 6 4 7900|
|                |  |    - 422 435 401 527 245   -   -   - 456   - 541 542   -   - 521 312 401 553   -   -   - 521 521 509   -   -   - 312 401 |    0 011845|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                |15|   26  42   -   3   X   X   X  11 ^12  13   -   X   X  53 ^54  20  13   X   X   X  11 ^12   9  14   X   X  25 ^26   9   X | 13 4 4 7906|
|                |  |  616 625   - 611   -   -   - 157 503 539   -   -   - 334 334 520 539   -   -   - 157 503 503 410   -   - 527 245 503   - |    0 012701|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                |16|   31 ^32  13 ^12   X   X   X  30   -   9   X   X   X  27  32 ^33  25 ^26   X   X  55 ^56  20   8   -   X   X   X   X   X | 15 6 5 7109|
|                |  |  420 420 225 503   -   -   - 542   - 503   -   -   - 703 553 135 527 245   -   - 703 110 520 800   -   -   -   -   -   - |    0 011528|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                |17|   27 ^28   9   X   X   X   6  ^7  27   -   X   X  32 ^34  51   X   X   X  18 ^19  50   -   X   X  28 ^29  13   X   X   X | 15 2 5 7107|
|                |  |  342 334 415   -   -   - 312 401 703   -   -   - 553 135 805   -   -   - 329 403 759   -   -   - 200 637 539   -   -   - |    0 011517|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                |18|   41  33 ^34   -  13   X   X   X  25 ^26  27  36   X   X  25 ^26  27   -   -   X   X   X  18 ^19   8   X   X   X   X   X | 16 8 4 7101|
|                |  |  347 606 345   - 539   -   -   - 527 245 703 542   -   - 527 245 703   -   -   -   -   - 329 403 800   -   -   -   -   - |    0 011404|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                |19|    -   X   X   X  24  11 ^12   X   X   X   1   6  ^7   X   X  32 ^33   4  ^5   X   X   X  24  10  30   -  43   X   X  10 | 15 4 4 7101|
|                |  |    -   -   -   - 550 157 503   -   -   - 521 312 401   -   - 553 135 251 735   -   -   - 550 509 542   - 553   -   - 509 |    0 010931|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                |20|   29   -  18   -   X   X   X   9  24   X   X   X   X  28 ^29   X   X   X  11 ^12  21  20   X   X   X  32 ^35  50  30   - | 16 3 3 7129|
|                |  |  648   - 559   -   -   -   - 503 550   -   -   -   - 200 637   -   -   - 157 503 608 520   -   -   - 553 110 759 542   - |    0 011121|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                |21|    X   X   X  28 ^29   2   -   X   X   4  ^5  20   X   X   6  ^7   8   X   X   X  13   -  11 ^12   2   X   X   X  32 ^33 | 15 6 5 7105|
|                |  |    -   -   - 200 637 541   -   -   - 251 735 520   -   - 312 401 800   -   -   - 539   - 157 503 541   -   -   - 553 135 |    0 011041|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                |22|   33 ^34   X   X   6  ^7   -   4  ^5   X   X   X  24  45   3   X   X   X  25 ^26  39  30   X   X  25 ^26   -   X   X   X | 15 3 5 7100|
|                |  |  606 345   -   - 312 401   - 251 735   -   -   - 550 330 611   -   -   - 527 245 553 542   -   - 527 245   -   -   -   - |    0 010948|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                |23|    X   X   X  15 ^16  43  44   X   X   X  17  10   -   X   X   X  32 ^33  20   -   X   X   6  ^7   3  24   X   X  18 ^19 | 15 6 4 7102|
|                |  |    -   -   - 436 334 553 720   -   -   - 456 509   -   -   -   - 553 135 520   -   -   - 312 401 611 550   -   - 329 403 |    0 011243|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                |24|   23  12   -   8   X   X   X  20   -  14  13   X   X  11 ^12   9   3   X   X   X  25 ^26   3  20   -   X   X   X   X   X | 16 6 2 7109|
|                |  |  503 500   - 800   -   -   - 520   - 410 539   -   - 157 503 503 611   -   -   - 527 245 611 520   -   -   -   -   -   - |    0 011526|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
|                |25|    X   X   X   X  11 ^12  25 ^26   X   X   X   9   9  52  10   X   X   X  28 ^29  20   -   X   X   X  41  24   2  10   - | 15 0 3 7107|
|                |  |    -   -   -   - 157 503 527 245   -   -   - 503 503 458 509   -   -   - 200 637 520   -   -   -   - 505 550 541 509   - |    0 010911|
|----------------|--| ------------------------------------------------------------------------------------------------------------------------ |------------|
```

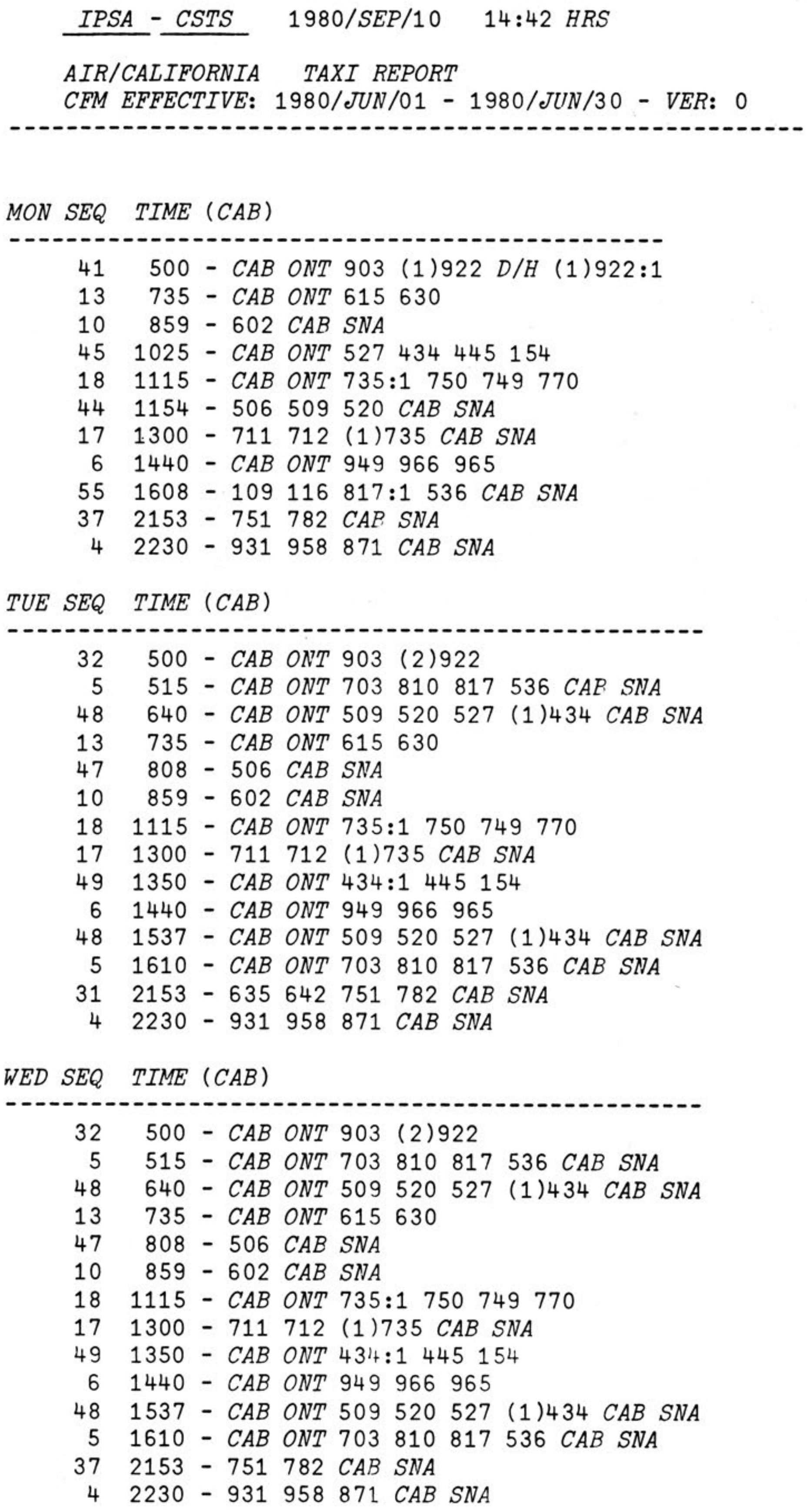
Figure 4


MANAGEMENT REPORTS
MARKET ANALYSES
OPERATIONS REPORTING
FLIGHT SCHEDULE TIMES
FLIGHT SCHEDULING
FLIGHT SCHEDULING PLANNING
DAILY CREW WORK ASSIGNMENTS
CREW TRACKING/ REPORTING
MONTHLY WORK SCHEDULES
CREW SCHEDULING
CREW PAYROLL SYSTEM



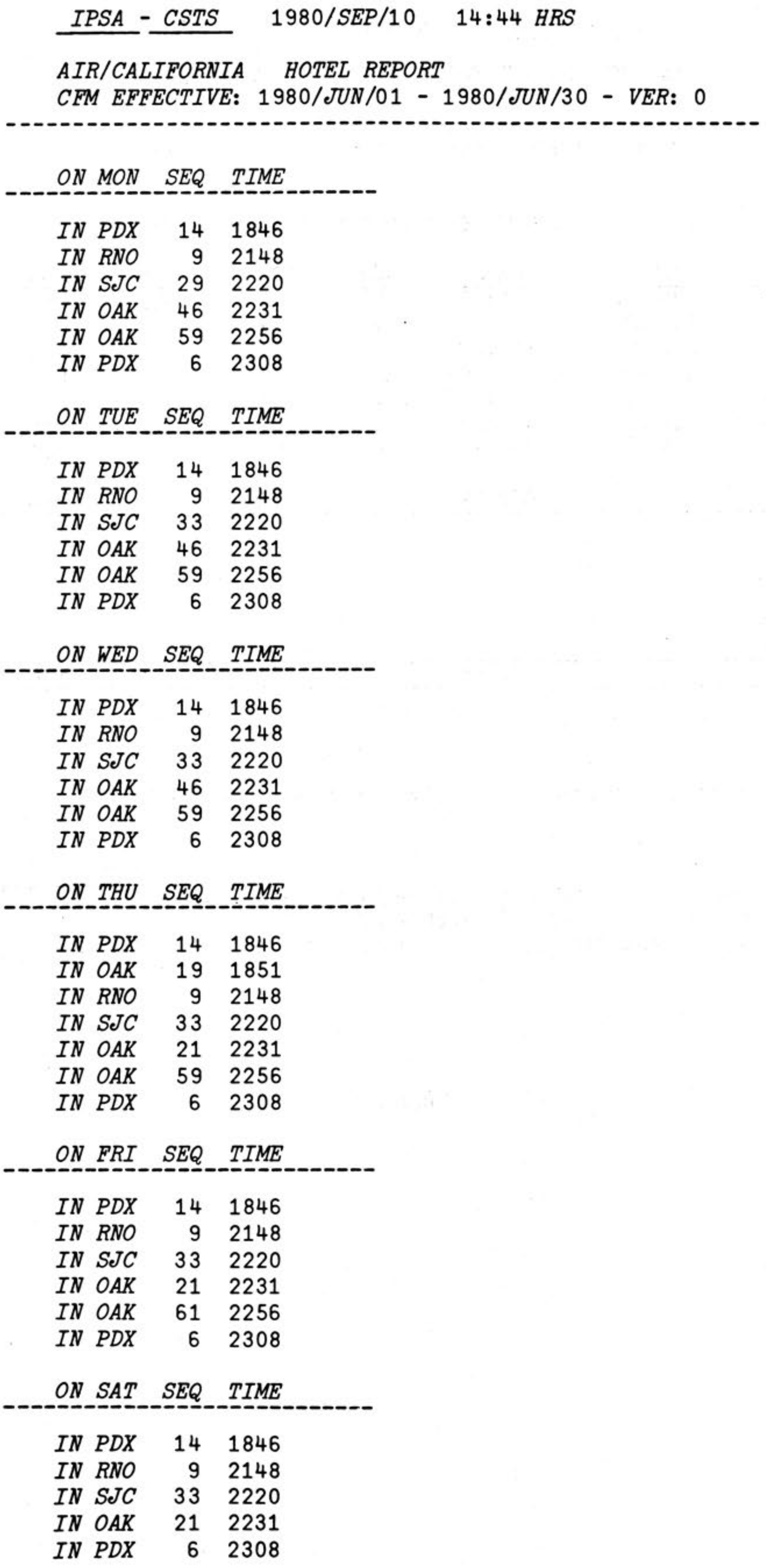
```
        PRINT SUMMARY 1980 6 1



  IPSA - CSTS     1980/SEP/10    14:41 HRS
 AIR/CAL   CREW FINDER SUMMARY EFFECTIVE: 1980/JUN/01 - 1980/JUN/30 - VER: 0

                              SEQ      OCC    SEQS     P/C       VALUE      o/o
DAY     SEQS     P/C         VALUE    MONTH    TOT    TOTAL      TOTAL      P/C
---     ----    -----       ------    -----   ----    -----     ------      ---

M        29      9:08       129:46      5      145    45:40     648:50      7.0
T        27     12:35       133:13      4      108    50:20     532:52      9.4
W        28     12:00       132:38      4      112    48:00     530:32      9.0
TH       28     13:11       133:49      4      112    52:44     535:16      9.9
F        28      7:30       138:11      4      112    30:00     552:44      5.4
SA       23      7:09       118:10      4       92    28:36     472:40      6.1
SU       25      6:20       119:56      5      125    31:40     599:40      5.3
---------------------------------------------------------------------------------

TOT     188     67:53       905:43     30      806   287:00    3872:34      7.4
---------------------------------------------------------------------------------
```

Figure 6



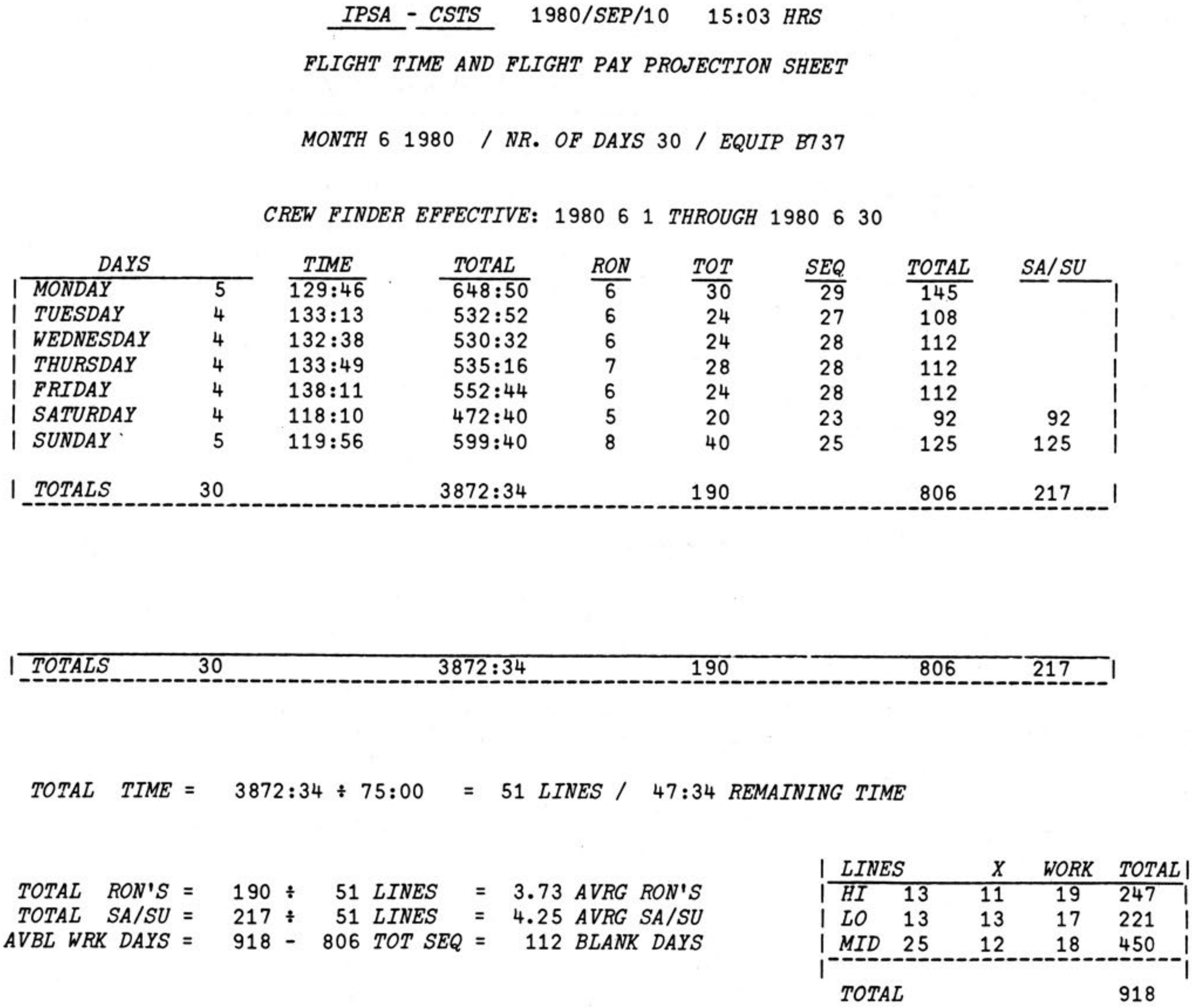
```
            IPSA - CSTS    1980/SEP/10   14:42 HRS

            AIR/CALIFORNIA   TAXI REPORT
            CFM EFFECTIVE: 1980/JUN/01 - 1980/JUN/30 - VER: 0
----------------------------------------------------------


MON SEQ  TIME (CAB)
------------------------------------------------
     41   500 - CAB ONT 903 (1)922 D/H (1)922:1
     13   735 - CAB ONT 615 630
     10   859 - 602 CAB SNA
     45  1025 - CAB ONT 527 434 445 154
     18  1115 - CAB ONT 735:1 750 749 770
     44  1154 - 506 509 520 CAB SNA
     17  1300 - 711 712 (1)735 CAB SNA
      6  1440 - CAB ONT 949 966 965
     55  1608 - 109 116 817:1 536 CAB SNA
     37  2153 - 751 782 CAB SNA
      4  2230 - 931 958 871 CAB SNA

TUE SEQ  TIME (CAB)
---------------------------------------------------
     32   500 - CAB ONT 903 (2)922
      5   515 - CAB ONT 703 810 817 536 CAB SNA
     48   640 - CAB ONT 509 520 527 (1)434 CAB SNA
     13   735 - CAB ONT 615 630
     47   808 - 506 CAB SNA
     10   859 - 602 CAB SNA
     18  1115 - CAB ONT 735:1 750 749 770
     17  1300 - 711 712 (1)735 CAB SNA
     49  1350 - CAB ONT 434:1 445 154
      6  1440 - CAB ONT 949 966 965
     48  1537 - CAB ONT 509 520 527 (1)434 CAB SNA
      5  1610 - CAB ONT 703 810 817 536 CAB SNA
     31  2153 - 635 642 751 782 CAB SNA
      4  2230 - 931 958 871 CAB SNA

WED SEQ  TIME (CAB)
---------------------------------------------------
     32   500 - CAB ONT 903 (2)922
      5   515 - CAB ONT 703 810 817 536 CAB SNA
     48   640 - CAB ONT 509 520 527 (1)434 CAB SNA
     13   735 - CAB ONT 615 630
     47   808 - 506 CAB SNA
     10   859 - 602 CAB SNA
     18  1115 - CAB ONT 735:1 750 749 770
     17  1300 - 711 712 (1)735 CAB SNA
     49  1350 - CAB ONT 434:1 445 154
      6  1440 - CAB ONT 949 966 965
     48  1537 - CAB ONT 509 520 527 (1)434 CAB SNA
      5  1610 - CAB ONT 703 810 817 536 CAB SNA
     37  2153 - 751 782 CAB SNA
      4  2230 - 931 958 871 CAB SNA
```

Figure 7



```
  IPSA - CSTS     1980/SEP/10    14:44 HRS

  AIR/CALIFORNIA    HOTEL REPORT
  CFM EFFECTIVE: 1980/JUN/01 - 1980/JUN/30 - VER: 0
-------------------------------------------------------------

  ON MON  SEQ  TIME
----------------------------

  IN PDX   14  1846
  IN RNO    9  2148
  IN SJC   29  2220
  IN OAK   46  2231
  IN OAK   59  2256
  IN PDX    6  2308

  ON TUE  SEQ  TIME
----------------------------

  IN PDX   14  1846
  IN RNO    9  2148
  IN SJC   33  2220
  IN OAK   46  2231
  IN OAK   59  2256
  IN PDX    6  2308

  ON WED  SEQ  TIME
----------------------------

  IN PDX   14  1846
  IN RNO    9  2148
  IN SJC   33  2220
  IN OAK   46  2231
  IN OAK   59  2256
  IN PDX    6  2308

  ON THU  SEQ  TIME
----------------------------

  IN PDX   14  1846
  IN OAK   19  1851
  IN RNO    9  2148
  IN SJC   33  2220
  IN OAK   21  2231
  IN OAK   59  2256
  IN PDX    6  2308

  ON FRI  SEQ  TIME
----------------------------

  IN PDX   14  1846
  IN RNO    9  2148
  IN SJC   33  2220
  IN OAK   21  2231
  IN OAK   61  2256
  IN PDX    6  2308

  ON SAT  SEQ  TIME
----------------------------

  IN PDX   14  1846
  IN RNO    9  2148
  IN SJC   33  2220
  IN OAK   21  2231
  IN PDX    6  2308
```

Figure 8



```
                            IPSA - CSTS    1980/SEP/10    15:03 HRS

                         FLIGHT TIME AND FLIGHT PAY PROJECTION SHEET

                         MONTH 6 1980  / NR. OF DAYS 30 / EQUIP B737

                      CREW FINDER EFFECTIVE: 1980 6 1 THROUGH 1980 6 30

      DAYS                TIME      TOTAL      RON     TOT     SEQ     TOTAL     SA/SU
 | MONDAY        5       129:46    648:50      6       30      29      145              |
 | TUESDAY       4       133:13    532:52      6       24      27      108              |
 | WEDNESDAY     4       132:38    530:32      6       24      28      112              |
 | THURSDAY      4       133:49    535:16      7       28      28      112              |
 | FRIDAY        4       138:11    552:44      6       24      28      112              |
 | SATURDAY      4       118:10    472:40      5       20      23       92       92     |
 | SUNDAY        5       119:56    599:40      8       40      25      125      125     |

 | TOTALS       30                3872:34             190              806      217     |
 ----------------------------------------------------------------------------------------




 | TOTALS       30                3872:34             190              806      217     |
 ----------------------------------------------------------------------------------------


   TOTAL   TIME =    3872:34 ÷ 75:00    = 51 LINES /  47:34 REMAINING TIME

                                                                  | LINES        X   WORK  TOTAL|
  TOTAL   RON'S =    190 ÷    51 LINES    =  3.73 AVRG RON'S      | HI    13    11    19    247 |
  TOTAL   SA/SU =    217 ÷    51 LINES    =  4.25 AVRG SA/SU      | LO    13    13    17    221 |
AVBL WRK DAYS =    918 -   806 TOT SEQ =    112 BLANK DAYS        | MID   25    12    18    450 |
                                                                  |---------------------------|
                                                                    TOTAL                  918
```

Figure 9

SHARP APL MULTIPROCESSING AND SHARED VARIABLES

Richard H. Lathwell
I.P. Sharp Associates Limited
Toronto, Ontario

Abstract

Facilities which have been recently introduced in SHARP APL, together with Shared Variables, permit an APL program to interact directly with the APL system. These facilities permit systems to be written in APL which use the APL system itself as a subsystem.

Introduction

A computer **system** is a collection of inter-related processes designed to achieve a specific set of tasks. Such a system can be represented by a collection of computer **programs**, each of which performs some function, and the orthography of these programs provides a **constructive definition** of the functions performed. A **processor** is a computing system which is capable of understanding such definitions and of carrying out the operations they describe. A **multiprocessing system** is a computing system which consists of two or more processors designed to cooperate concurrently to achieve a common objective.

The introduction of shared variables permitted certain kinds of multiprocessing systems to be described and constructed in APL by providing the ability for two concurrent processes to communicate by means of a variable whose name and value is known to both. The systems which can be described in this manner, however, consist of collections of **autonomous** processors: no one of which can explicitly direct the activity of any other. Shared variables only provide communication between processors; their associated access control is a mechanism to guarantee message delivery. In particular, an APL time-sharing system cannot be constructed, since there is no way in which the subordinate relationship of an APL processor to its environment can be implemented.

In 1973 [1], an APL processor was formulated in which external communication took place through a shared variable. A similar concept was later implemented by B.J. Hartigan of IBM [2], and this implementation was successfully used as a component in a system where terminal input and output was written and executed in APL [3].

This paper describes work-in-progress at I.P. Sharp Associates to provide a similar interface for SHARP APL. There were two primary motives for undertaking this work: a desire to explore the efficacy of a shared variable interface to the APL processor, and a desire to allow access to APL by terminals which are incompatible with the

SHARP Communications Network and with the existing terminal input-output interface.

APL Processor Characteristics

Most implementations of APL are interactive; all activity is directed by orders provided by a user. The resultant APL processor has the following characteristics:

- It waits in an idle state until input is received.
- It then processes the order represented by the input, perhaps producing output, then again awaits input.

Special facilities such as "break" or "attention" are provided so that the user can always force the processor into the input state.

For the purpose of implementing a shared variable interface in SHARP APL, the APL processor was formalized slightly more rigidly as follows:

- A **line** of input or output consists of a sequence of APL characters.
- The APL processor waits for a **single line** of input.
- The input is processed, and results in an arbitrary number of lines of input, but no fewer than one.
- The final line of output is an **input prompt**, and contains an indication that the APL processor is waiting for input.

It is thus possible to construct a shared variable interface to the APL processor whose values are character vectors which represent lines of input or output, and to devise a simple APL program which can direct its operation.

An APL system is a collection of one or more APL processors and one or more terminals. The processors can be active and communicating with a specific terminal, or they can be inactive with no terminal association. An essential function which must be provided by an APL system is that of activating an idle processor and associating it with a specific communication interface. The manner in which a user requests a processor varies widely among APL systems, from the simplicity of a power switch to the complexity of a system function, such as ⎕*RUN*, designed to perform this specific operation. Similarly, an APL system must dissociate and deactivate processors as appropriate.

SHARP APL S-tasks

APL processors whose input-output interface is a shared variable are designated, in the vernacular of SHARP APL, as S-tasks and are provided by two fundamental parts of the system. The first is the ability of the system to communicate terminal input and output via a shared variable; SHARP APL has been augmented to operate as an auxiliary processor (a program which is not written in APL but which can share variables with an APL program). The second is a controlling program which is responsible for the allocation of S-task processors and their associations with specific shared variables.

Establishing an S-task

When SHARP APL has been installed with appropriate configuration parameters, it connects itself to the shared variable processor as **processor 1** during system initialization. An S-task may then be created by offering to share a variable with processor 1:

```
      1 ⎕SVO 'NAME'
```

where `'NAME'` is a vector of 15 or fewer characters denoting the name of the variable to be used for the interface. When resources permit, the S-task controller will allocate the necessary tables, obtain a clear workspace, assign a task identification number, and then match the offer to share. When the degree of coupling becomes 2, the task is in the same state as a T-task after the terminal has become connected to the system, but before the user has signed on.

Communication with an S-task

S-task input and output is in the form of APL character vectors of four or more elements. The first four characters of each value contain control information which must be present and must be examined. The first two of these characters are not used at the present time, and must be binary zeroes, i.e., `⎕AV[0 0]`. (**Note**: index origin 0 is assumed throughout this discussion.) The third element is used to denote commands to the S-task controller from the initiator and responses from the S-task controller. The fourth element is used to indicate the meaning of the value which is being passed from SHARP APL to the initiator.

For example, the first input to an S-task will normally be an attempt to sign on to SHARP APL:

```
      NAME←⎕AV[0 0 0 0],')1234:KEY'
```

Assuming that the account number and key are valid, the response returned in *NAME* will be:

```
      prefix,'2134)22.13.00 04/01/80 SUEBLUE'
```

i.e., the normal indication of a successful sign on. Otherwise the response might be:

```
      prefix,'INCORRECT SIGNON'
```

or

```
      prefix,'NUMBER NOT IN SYSTEM'
```

The important point to note is that, for every value set in the interface variable, APL or the S-task controller will respond with one or more values in reply.

Qualification of Values Received From APL

The fourth element of character vectors received from APL indicates the meaning of the rest of the vector. If

```
      REPLY←NAME  ⍝ RECEIVE VALUE FROM APL
      MODE←(8⍴2)⊤⎕AV ⍳ REPLY[3]
```

then the elements of *MODE* have the following meaning when their values are 1 (and the converse when 0):

MODE[0] - The S-task workspace is in immediate execution mode, i.e., is in a state where system commands will be recognised and processed.

MODE[1] - The rest of the vector contains an obfuscating blot to obscure passwords, etc.

(*MODE*[0 1] occur in conjunction with *MODE*[6]).

MODE[2] - The S-task is not signed on to SHARP APL.

MODE[3 4] - are not used and are always zero.

MODE[5] - The vector contains arbitrary output from ⎕*ARBIN* or ⎕*ARBOUT*; either *MODE*[6] or *MODE*[7] will also be 1.

MODE[6] - The value is a prompt for input: SHARP APL expects input after this output. Normally, the input will be catenated to this prompt by the initiator, and returned to APL. However, the initiator may instead process the prompt in conjunction with additional input, and return the processed information to APL.

MODE[7] - The value is output, and more output will follow.

S-task Controller Commands

The third element of values set by the initiator is a command to the S-task controller as follows:

⎕*AV*[0] - indicates the character vector is APL input.

⎕*AV*[1] - is a request for accounting information. The response from the S-task controller is a three-element **integer** vector containing 3↑⎕*AI* of the S-task. If the S-task is not signed on to APL all three elements will be zero.

⎕*AV*[2] - is a request to signal BREAK, analogous to depressing the BREAK key on a terminal connected to a T-task.

⎕*AV*[3] - is a request to permit retraction of the interface variable. If this status has not been requested, retracting the variable will result in the immediate termination of the S-task in a manner similar to the termination of a T-task when the telecommunication line is disconnected. The interface variable will remain as an offer from processor 1 to the initiator; resharing resets the permission to retract.

⎕*AV*[4] - is a request to reset the retraction permission obtained previously by ⎕*AV*[3].

S-task Controller Responses

The S-task controller responds to each non-zero command (except for accounting information which returns an integer vector) with a four-element character vector whose third element indicates the result of the command:

⎕*AV*[0] - indicates that the command was successfully executed.

⎕*AV*[1] - indicates an invalid value: an invalid command value, non-character value, non-vector value, or a character vector of fewer than four elements.

⎕*AV*[2] - indicates that APL could not accept input because of insufficient workspace.

⎕*AV*[3] - indicates that the S-task controller refused to honour a request because the S-task was not signed on to SHARP APL.

S-task Termination

An S-task will be terminated by any of the following:

- Sending a valid signoff command, e.g.,)*OFF*, when *MODE*[0] is 1.
- Retracting the interface variable, e.g., ⎕*SVR* '*NAME*', without disconnect permission.
- A request by the S-task for input when the interface variable is retracted with disconnect permission.

The S-Task Development Project

The project to implement SHARP APL S-tasks has been an interesting experience for those involved, partly because the entire facility was programmed in APL. The first step was the development of a detailed APL model of APL input and output, including function definition and editing, using arbitrary input-output facilities for the actual terminal communication. When this model behaved satisfactorily, the ⎕*ARBIN* and ⎕*ARBOUT* uses were replaced with a shared variable interface so that the model could be driven from another workspace which contained an APL model of terminal-supporting software. When this two-processor model was satisfactory, the process of modifying the APL system itself began.

At the same time, Eric Iverson undertook to develop support for the IBM 3270 terminal and began by writing the support as a working APL model, controlling the device by a very simple auxiliary processor. This model was evaluated in conjunction with the S-task model, resulting in a number of important design decisions (for example, the input-output prefix conveying state information) to be made and tested in a very short time. This model was then translated to assembly code.

Each part of the project was then able to proceed independently, using the APL model of the other for testing, analysis, and evaluation, since the assembly language interfaces and the APL model interfaces are identical. This ability to program independently once the interface specifications had been agreed vastly reduced the elapsed time required for testing. If all programming had been done in assembly language, testing would necessarily have been sequential. The total programming time was probably also reduced (we don't know, we don't have time to do it the other way and then measure the difference) because most of the revisions of specifications were tried in the APL models first; almost no assembly-code revisions were required.

Conclusion

Although the S-task project is not yet complete, it has produced a prototype working sufficiently well for testing at a customer installation. Evaluation of the APL models and of the prototype suggests the following:

- It is advantageous to represent the APL terminal input-output interface as an APL object because such an object provides a well-defined machine-independent interface which can be manipulated by APL programs.

- Hence, it possible to implement terminal supporting code directly in APL.

- The most important implication is that any program which can share variables has the ability to use the entire SHARP APL system as a subsystem.

References

1. Lathwell, R.H., **System Formulation and APL Shared Variables**, IBM Journal of Research and Development and D, Vol. 17, No. 4, July 1973.

2. Hartigan, B.J., IBM Internal Publication, 1976.

3. Lathwell, R.H., IBM Internal Publication, 1977.

PROGRAMMER PRODUCTIVITY GAINS THROUGH EMPHASIS OF FUNDAMENTALS

Jon McGrew
IBM Corporation
Kingston, New York

Abstract

Many papers have been published which deal with APL programming productivity, but many of these approaches are aimed at the programming professional who is knowledgeable enough to be able to coax the last gram of performance out of the machine. Unfortunately, in these do-it-yourself days of APL programming, most part-time programmers will simply never get that far, and user productivity may be far more urgent than machine productivity. While "fine-tuning" an application may result in measurable savings, there are many more fundamental approaches which can yield huge benefits in both programming time and machine time. This paper simply re-emphasizes these fundamentals.

* * *

If you do any APL programming at all, you've probably seen it happen: you spend a month putting an application together, and then show it to another APL programmer, who says, "Oh, I can write a better one." And he proceeds to do just that, from scratch, in two evenings of spare time. Does he **really** know a hundred times more about APL programming than you do? Probably not. But he probably knows a few tricks of the trade that can save an immense amount of time, and improve the final results.

APL has always enjoyed a reputation as a do-it-yourself programming language; a language which is easy and quick enough to work with that many non-programmers write their own applications rather than farming them out to a programming group. That's fine, but these part-time programmers are often the ones who can benefit the most from a few shortcuts.

"What's this beginner's stuff doing here at the conference?"

Unlike most of the papers at this conference, this paper is aimed at the beginner. Why? Because publication of this type of information to the users of our own system indicated an overwhelming thirst for ground-level information, and the conference is one means of distribution of ideas to the beginning users of other systems. This paper may also be considered to be an "opposing viewpoint" to those who would spread the word of programming efficiency by means of substitution of functions to produce miniscule savings, when so many users need thoughts on a basic approach. This paper, therefore, isn't aimed primarily at the programming professional, but rather at the hundreds of occasional programmers who comprise a substantial part of the population of so many of the APL systems.

Developing A Building-Block Approach

What would it be worth to you to find a way to write functions more quickly and easily, make trouble-shooting considerably faster, aid documentation, **and** prevent *WS FULL* problems, all with one technique? ... Read on, Macduff ...

A "building-block" or "modular" approach to writing APL functions is a writing style in which the main function calls sub-functions to perform the details of the work, rather than trying to put all of the code into the main function. Developing an APL writing style like this can result in some rather substantial benefits for you.

Answering lots of phone calls from our customers as I do, I see quite a bit of programming written by people with a wide variety of backgrounds and approaches. One distressing sight that I often see is a workspace containing an extremely lengthy function with no subroutines. Now I realize that most of the users of my system aren't programmers, but it's sad to see APL functions like this because substantial savings could have been realized both in original writing time and in subsequent trouble-shooting if the author had used a building-block approach in writing his application.

Probably each of you have seen examples like the ones here; probably many of you also have some similar ones of your own (I have lots of them, but I'm fixing them little by little). First, take the case of this portion of a billing function that I once worked with:

```
[65] PRTOT1:Y← 0 60 60 ⊤TT[4;0]
[66]  Z← 0 60 60 ⊤(TT[4;1]×3.33)÷1000
[67]  CON←(4 0 ⍕Y[0]),':',(2 0 ⍕Y[1])
[68]  →(0=∨/S←CON[¯6↑⍳7]=' ')/L20
[69]  CONS[S/¯6↑⍳7]←'0'
[70] L20:CPU←(3 0 ⍕Z[0]),':',(2 0⍕Z[1]),':',(2 0 ⍕Z[2])
[71]  →(0=∨/S←CPU[¯8↑⍳9]=' ')/L21
[72]  CPU[S/¯8↑⍳9]←'0'
[73] L21:OUTPUT←TEMP[CHG],'  ',NAME,' ',CON,'  ',CPU, 10 0 ⍕TT[4;REST]
[74]  PRT OUTPUT
```

Now don't get me wrong, this function does work. It's just a little obscure. An example of opacity in coding (perhaps the real meaning of "coding"). However, a reader of the function doesn't normally need to know minute detail of any portion of the code until he identifies the portion that needs repairs. This can easily be achieved by simply enclosing pieces of the function in sub-functions, whose contents are unimportant until we need to see their own particular piece of the operation:

```
[21] PRTOT1:CON←FORMAT RATES[0]×TT[4;0]
[22]  CPU←FORMAT RATES[1]×TT[4;1]
[23]  PRINT TEMP[CHG],'  ',NAME,' ',CON,'  ',CPU, 10 0 ⍕TT[4;REST]
```

Notice that this function is notably shorter than the previous one. The original one was 118 lines long (ugh!). How long **should** a function be? Well, Abraham Lincoln observed that a person's legs are the right length when they just touch the ground. In the same spirit, there's no particular rule that says how long a function can be. But if you properly observe building-block techniques, you'll probably find that your average function length is pretty short.

And what's that? Well, I pondered this for quite a while. As Supreme Court Justice

Potter Stewart noted when questioned on programming style, you'll recall that he answered that "I may not be able to define pornography but I know it when I see it."

Not wanting to have him look at **my** workspaces just yet, I hurried up and checked them myself. Here's the tally: distributed through 12,366 unlocked functions were 145,090 lines of code, giving an overall average of 11.7 lines per function (counting comment lines). The average width, by the way, (excluding comment lines) was 19.2 characters.

Keeping the average to under twenty lines insures that you will be able to display and edit the function much more easily on video terminal, and you'll save a lot of printing time on a hard-copy terminal. Also as a rule of thumb, any one single function **should always** fit on an 8½ x 11-inch sheet of paper when it's displayed. Of course, Lincoln **also** observed that "important principles may and must be flexible," but if an APL function is longer than a page, you're **probably** trying to accomplish too much with one function. See the section on abstracts ("Where's This Function Going?") for more on this.

1. Save Yourself Writing Time

This benefit comes in several flavours. First, a short function is very easy to display and edit. Imagine having to repeatedly display that 118 line function during its development. Second, there is a finite amount of material that you can concentrate on at one time. Keep a given function focused on a specific goal. In that way, **you** can also concentrate on that same goal without being drowned in extraneous details. And third, once you identify the specific purpose of the function, you may not even have to write it at all! What's that you say? Very simple...

2. Become a Friend of the Public Library

Take advantage of the voluminous resources of the APL Public Libraries available on many systems. If you're trying to accomplish some very standard operation, why re-invent the wheel? While perhaps no one else has ever written a package just like the one that you're working on, certainly **many** of the bulding-blocks will be the same. You may have designed your own very unique house, but chances are you didn't have to design the bricks, lumber, nails, and wiring components. Why do that with your APL code?

The classic analogy to making your own nails is writing a lengthy *REPORT* function which spends several lines, labels, and branches just formatting the current time and date to print on the report. What a waste of time for an author to spend his good time figuring that stuff out, when his real goal is to produce a specific report. Does he assume that he's the first one who has had to timestamp a report? Building-blocks like this are available in abundance in many Public Libraries, free for the taking.

When you're writing an application, your goal is the overall operation of that package. You shouldn't have to spend the time concerning yourself with minute details that have certainly been solved before by others. By using Public Library functions, not only are you saving yourself from re-inventing the wheel, but you are using building-blocks that are often better than the ones that you would have been able to build yourself. Keep in mind that the author of a workspace containing building-blocks had **those blocks** as his goals, and therefore probably had the time to consider (and solve) many problems that you might not have thought of.

3. Simplify Your Trouble-Shooting

Debugging any program can become a tedious task; anything that helps to lessen that task is a welcome treat. A building-block approach immediately reduces the task. In the aforementioned billing function, let's suppose that the connect and CPU times were printing improperly. In the first example, you may have had to print over seventy lines of code to find the code that was causing the problem, and even then it would have required making the same change to several sections of code. Using building-blocks (and assuming that you have used obvious names for the blocks), you or anyone else should be able to quickly locate the culprit. With the sub-functions in the second example, changing the *FORMAT* function (which is perhaps five lines long) could solve the problem; you wouldn't have to be concerned with the main function at all. And changing that one function would fix the problem wherever the function is called.

You'll find that a modular approach to functions will immediately put at your disposal the considerable resources of (again) the Public Library for debugging aids. There are many packages offered there which will analyze the interaction of functions without being very concerned with what's inside those blocks.

4. Simplify Your Job of Documenting Your Application

Documentation is usually considered to be most programmer's anathema. There are several forms of documentation. One is in the form of a formal manual. Another is comments within the code. Another is the code itself. Compare the two examples that we have discussed so far. Certainly the first function would be the easier one for the reader of its user's guide to correlate to the text. The number of comments required in the code are drastically reduced if appropriately named sub-functions are employed, since the code itself becomes much more self-documenting.

5. Prevent *WS FULL* Problems With Building-Blocks

The original billing function shows some identical lines of code being repeated. Everything that gets entered in the function uses space that can't be used for something else. Using a sub-function allows one copy of the code to be used several places in the workspace without taking several times its storage space. And, by properly localizing variables you don't have to take any special care to expunge variables that you're through using; they'll go away on their own, thereby freeing up the space that they were consuming.

* * *

Considering the rather substantial benefits that can be realized by using a building-block approach, it's really a shame that more people aren't taking advantage of it. Don't cheat yourself; give it a try. Start by making a conscious effort to use the technique. Chances are, you'll soon have altered your writing style to the point where you'll wonder why you didn't always write that way.

6. Some Considerations in Writing Building-Blocks

A building-block should represent a single isolated operation. It should be something that can be commonly used, usually without any modification, by many applications. For example, a *REPORT* function that reads the contents of one of your datasets and breaks your data down into sub-totals is **not** a building-block. The function that opens the file **could** be a building-block, if sufficient thought is given to it so that it can also be used elsewhere. The same could be said of the function that reads each block of data from the file... or of the function that checks the return code from an auxiliary processor. There's no reason to burden the main calling function with specialized code to do this sort of thing when you're going to have to do the same thing with **many** applications. Write it at one time, and be done with it.

So how can a function be generalized? Well, to be considered a building-block, a function almost certainly has to use arguments and an explicit result. If a function doesn't have arguments, where is it getting its data? The use of global variables makes for very obscure applications. And if the function doesn't have an explicit result, it can't pass data along to another function for continued processing.

An example of a real gound-level building-block function (a real **nail**) is a function called *DMB*. This function is designed to "delete multiple blanks", by dropping all leading and trailing blanks, and reducing multiple contiguous blanks within a character string to single blanks.

```
      L←'     THIS    IS   IT   '
      ⍴L
21
      DMB L
THIS IS IT
      ⍴DMB L
10
```

This function is so generalized that it can be considered to be almost a primitive; it just happens to be written in APL. This function is very useful when you wish to pull a line out of a character matrix and use it elsewhere; *DMB* lops off all of the unneeded blanks. It is also quite helpful when prompting for input with ⍞. Using *DMB* means that you don't have to be concerned if you get back a few extra blanks along with the response (as happens with "bare output", for example).

Another tiny building-block is a function called *THRU*... `3 THRU 7` yields `3 4 5 6 7`. It's obviously not related to any particular application; its just a useful tool for saving yourself a little bit of effort.

Most of the building blocks that you put together may well be more complex, and therefore more specialized, than these examples. But remember that the more generalized you can make the function, the higher the probability is that you will be able to use it again in another application.

You will undoubtedly find that some things that you use building-blocks for early in your APL career may later on be discarded in favour of just entering their definitions directly. For example, where you used to use a function for right-justifying a matrix of text...

```
      M←RIGHT∆JUSTIFY M
```

... you might just replace later with its definition:

```
      M←(-+/∧\⌽' '=M)⌽M
```

"What gives? I thought you just tried to talk me into using building-blocks?" True, true. But building blocks may **also** serve the purpose of providing education. If you haven't had to left- or right-justify a matrix of text before, having a small stand-alone function to experiment with is certainly much more convenient than trying to observe that one line of code within a large application. If that expression is something that you use several places within one workspace, then it should remain as a separate building-block. If it's used only once in the workspace, but you don't know how to code it from memory, then a building-block would also make sense. But if it's only going to be used once in the workspace, if you know how to code it, and as with the above example, if the actual working code is no longer than its name, it may be just as easy to code it directly.

"Can't that be carried to extremes?" ... Sure, most things can. Occasionally we see an application that has a set of defined functions that look like this:

```
     ∇ Z←A PLUS B
[1]    Z←A+B
     ∇
```

The authors of these functions have possibly missed the point of modular approaches... they aren't making the overall function any easier to read or maintain, and they aren't saving space by reducing multiple definitions. Don't break things down this far.

Precisely where the divisions should occur, we can't say. We can give some examples, but there are no rules. A big factor in how far you break them down is your own degree of familiarity with APL.

Building A Toolbox

Have you ever seen a really devoted home handy-man with an extensively-equipped wood-work shop? These guys seem to be able to whip up a beautifully-made walnut coffee table in the time that it would take me to gouge out my own specialty (an ash tray), using my trusty (rusty) combination screwdriver/chisel. So how do they do it? Obviously, part of it is careful training and skill. But an equally important element is having a well-equipped tool-box, knowing where all of these tools are, and knowing how to use each of them the right way.

The same idea holds true with APL programming... or certainly, with nearly any undertaking. If you want to build big complex packages quickly, you need to have a well-equipped tool-box. In APL terms, that means having access to a workspace (or workspaces) that contain some basic building-blocks. These blocks can then be used repeatedly as elements of many larger applications.

When you first get into APL programming, things are very slow; you're often left with the feeling that in order to do anything, you have to concern yourself with all the details of the universe. But as time goes by, you may realize that many of the components of a new application that you're working on are really identical to the same blocks in your last application. And so those don't need to be written. The longer you work with it, the more building-blocks you have at your disposal; hence, the faster, easier, and more pleasurable the work becomes... more pleasurable because you can concentrate your efforts on the truly creative aspects of each new job, rather than being buried by the drudgery of low-level details.

In the section on "Building-Blocks", we recommended that you gather subroutines from the Public Library rather than writing your own. Now we're not suggesting that using any Public Library workspace will solve all your programming problems and save you from having to write any common subroutines. Nor are we suggesting that all of the functions in any given library will be useful to you. But it's a starting point. Take the ones that seem useful to you, and add to them as you develop ones of your own. Others will come from friends or perhaps from examples in textbooks.

Where Did This Function Come From? and Where Is It Going?

1. First, Where Did This Function Come From?

In the section on creating "building-block" applications, we made it sound like all you had to do was put things in small packages and you're all set. Well, there's one obvious problem that arises. How do you keep track of all those common subroutines? Let's assume that you have gotten some small functions from scattered sources; some from friends, some from the Public Library. Maintaining enough records to be able to get back to the same author to find other functions (or to fix problems with the existing ones) can be a problem.

To compound the problem, when you get many applications that have used many of the same subroutines, finding out which one is the most recent copy is often tedious. But hold on; there's a simple solution (did you **really** think we were going to say there isn't any solution?).

When you write a function, a very good habit to get into is to record the date that it was written as a comment within the function. Also quite useful is having the date of the last update. Finally, an invaluable piece of information is the identity of the author.

Human nature being what it is (and programming schedules being what **they** are), you are probably just **not** going to edit these dates each time that you make some minor change to the function. But if you don't change them, the whole point of the dates is lost. Therefore, what's needed is a very simple method of letting an APL function make those changes for you. We have done this on our own projects by using what we've come to call the *FIX* function.

When any function is first started, the first (by now, automatic) thing to enter is today's date as a comment on line one, using what for us has become a standardized format:

```
      ∇REPORT
[1]   ⍝10/03/80
```

After we're done entering the body of the function, the *FIX* function is used to update that date line to include the date of last update (which will be the same now, but will change in the future), and the name of the author:

```
      FIX 'REPORT'
⍝10/03/80 → 10/03/80 MCGREW, 63C, KINGSTON
```

The *FIX* function shows the newly formed line (just for a bit of feedback), and also puts this change in line 1 of the function. The function now looks like this:

```
      ∇REPORT[⎕]∇
    ∇ REPORT
[1]   ⍝10/03/80 → 10/03/80 MCGREW, 63C, KINGSTON
[2]   ...
```

Notice again that the creation date (the first one) and the date of last update (the second one) are the same for this function, because we just started to write the *REPORT* function. After changing another older function, it would look like this:

```
      FIX 'OLDCODGER'
⍝04/15/72 → 10/03/80 MCGREW, 63C, KINGSTON
```

Letting the *FIX* function enter the information for us rather than typing it in results in a couple of advantages: It provides a rigid format which will be the same in all of the functions. This means that those functions can be then examined under program control to find functions which were updated since a particular date, and so forth. And more important, having an easy means by which to timestamp functions makes it a little more likely that they **will** get timestamped!

By the way, putting your name in the function does more than just help some other user get back in touch with you when he has a problem with that old jerry-built function that you gave him. It also gives proper credit where it's due for the other superb functions that you gave him. And (only slightly less subtle), it may help to instill some further pride of workmanship in a well-written function. ...Always produce things with enough pride that you're happy to sign them.

2. Where's This Function Going?

It may seem obvious that you should decide what each APL function is supposed to do **before** you write it, but this fundamental seems to get overlooked rather often. We've all seen examples of functions which just seem to ramble about aimlessly, without any discernable purpose in life. Roget's Thesaurus shows "purpose" as a suitable synonym for "function" (of course, realize that it also shows "gathering"... and **that** one may be more suitable for some of the functions we've seen). Do each of your own APL functions have a single specific purpose?

Refer also to the section on "Building-Blocks" for a discussion of creating a connected series of "black-boxes". When you begin to write a function, the first thing that you should do is decide what **one** particular operation you wish to perform, and document that with the comment line in the beginning of the function **before** you write any code. This line is called an "abstract" (and hopefully prevents the rest of the function from becoming the abstract portion). The abstract line shouldn't ever exceed 60-80 characters. You may find that you can't state the function's purpose in just one line. Stop! Don't just write a longer one! That's probably a good indication that you're trying to take too big a bite with that function ... narrow its scope a bit and get it down to one operation before you start to write **any** code.

After you are into the writing of the function, this comment line will give you a good reference point to be sure that you aren't wandering from your purpose. You'll be surprised how much time you can save in writing the function by taking a minute to consider the scope of the function before you start, and clear your mind of any side purposes ... leave them for the next function.

Take pity on the programmer who will eventually inherit the maintenance responsibility for this application. How does he learn what it's comprised of, and what each of the building-blocks do? If each function contains a one-line abstract, it's rather easy to write a function that simply displays each of the function names along with their corresponding descriptions. In this way, the workspace could be made to explain itself. The person who has to maintain the application will be able to understand the purpose of the function by reading just one line, even though the rest of it may be completely obscure to him. He could then easily replace any function with a functionally equivalent one which is more obvious or runs faster.

And remember, that person who has to go back and modify this strange beast two years from now, **may** be **you**.

Looping Versus Using Arrays

Part of the real power of APL is its ability to deal directly with arrays, without having to be concerned with details of each element. Frederick J. Brooks explains it with his apple story. If he wants to have some apples brought up from the barrel in the basement, he just asks his son to bring up some good apples from the basement... he points out that he **doesn't** say, "Son, first, go down to the basement. Then, go to the apple barrel. Pick up an apple and look at it. See if it's a good apple; if it's not, set it aside. If it is, set it aside in a different spot so that it can be brought upstairs. Then, pick up another apple and look at it..." Professor Brooks of course included the proper checks to insure that his son didn't keep checking long after he's depleted the supply of apples.

And so it is with APL. One needn't be concerned with lots of nuts-and-bolts details; the APL designers have done an admirable job of hiding all of that stuff beneath the surface. If you have a thousand numbers stored in an array called *TABLE*, and you choose to add two to every element, *TABLE*+2 or 2+*TABLE* is all that's needed. Show a table of numbers to any ten-year-old, and tell him that it is called *TABLE*, then ask him to write down the arithmetic for adding two to each number.

It seems sad, then, with APL working in such straightforward ways, that so many people insist on writing loops into all of their code. I recently saw this example (Bad Example Number One), not once, but many times in one function:

```
ARRAY[1;8;]←MATRIX[1;]
ARRAY[2;8;]←MATRIX[2;]
ARRAY[3;8;]←MATRIX[3;]
ARRAY[4;8;]←MATRIX[4;]
ARRAY[5;8;]←MATRIX[5;]
ARRAY[6;8;]←MATRIX[6;]
```

In this example, *ARRAY* is a three-dimensional array, although the only row that we are changing is row 8. It is being fed some data from *MATRIX*, which has the same number of columns as *ARRAY* has. And, the number of planes in *ARRAY* is the same as the number of rows in *MATRIX*; in other words, their first dimension match.

This isn't meant to be a reflection upon the person who wrote the function. He was simply using the background that he had acquired in using other languages, and perhaps didn't realize how much effort he could have saved by going to simpler approaches. Now, unfortunatley, the "simpler" approaches that I've seen some people resort to is to remove that "inline" code, and replace it with a loop, like this:

```
 C←0
 H←1↑MATRIX
LP:C←C+1
 →(C>H)/0
 ARRAY[C;8;]←MATRIX[C;]
 →LP
```

Bad Example Number Two does have some advantages over Bad Example Number One. For instance, since it checks the height of *ARRAY*, notice that it will work regardless of how many planes there are in *ARRAY*. Well, that's progress; at least the author wouldn't have to add more lines of code if the size of the array is increased some day.

However, all that's needed to accomplish that same task is one concise expression:

```
ARRAY[;8;]←MATRIX
```

I figured that I'd better set this one off from the text the same as the other two examples, or everybody would go skipping right past it.

Yep, that last example does just the same thing as the first two examples... but it does it much quicker, and with many fewer keystrokes for the author. And consider the job that APL has to go through to execute any of these functions.

One of the frequent criticisms of APL is that it is "interpretive", that is, each symbol that you key in has to be separately resolved by APL before it can do any useful work for you. An alternate approach is a "compiled" program, in which the compilation step sets up internal pointers to the data and operations and allows that program to run very fast. Taking advantage of arrays, however, gives you sort of the best of both worlds. Some of the computer time that you use is spent resolving the interpretation of the names and symbols, and some of it is spent actually manipulating your data. Obviously, we would like the first time to be zero and the second time to be 100%. **If** the number of names and symbols that you use is small, the interpretation is very quick, and the bulk of the execution time is expended toward your end product. The internal coding for each of the APL primitive functions has been written in Assembler Code by very knowledgeable system programmers, and most have been optimized for lots of special cases. Doing things in big steps with arrays instead of little steps with loops allows most of the data movement to be handled by these optimized modules. In many cases, you'd be hard pressed to beat the speed even if you were writing your own programs in Assembler Language instead of APL, because the APL modules have years of optimization behind them.

As an example of how significant the gains in using arrays can be, consider the following examples:

```
SCALAR←2
VECTOR←1000ρ2
```

We constructed several expressions, ran them each 100 times, and measured the average execution time. "*A←SCALAR+SCALAR*" took `0.2` milliseconds, while *A←VECTOR+VECTOR* took `1.0` millisecond. That says that we can add a thousand numbers to a thousand other numbers in just five times as long as it takes to add two numbers together (`2+2`). Almost all of the time for the *SCALAR+SCALAR* case is used for interpretation.

Projecting that out a bit, that says that if we added those two scalars together a thousand times (to equal the work done by *VECTOR+VECTOR*), it would take 200 milliseconds... two hundred times as long as the array operation. Care for an easy 200× speed-up for your function?

But wait, even that isn't the whole story... If you are adding two numbers together a thousand times, it's unlikely that you actually have 1000 identical lines of *SCALAR+SCALAR* in your function; you're probably using that old devil loop again. And, of course, each of the symbols that comprise the loop have to be interpreted 1000 times. Trying this, we came up with an average time of 892.3 milliseconds. That's 892 times longer than the *VECTOR+VECTOR* case, and the result is the same.

Where else can you get an 892× performance improved by removing code?

* * *

Naming Conventions

Naming conventions always get a lot of press in the programming "standards" documents that we've seen from various applications groups. Without dwelling on the myriad individual rules that various groups have ssen fit to enforce, herewith are the two rules that we have found to be the most helpful.

1. Using Standardized Prefixes On Names

A helpful practice to follow is to name all of your datasets that pertain to a particular project with a standard prefix, so that they are grouped together in sorted listings. This practice, of course, can also be put to use with workspace names, so that similar workspaces are grouped together in the *)LIB* list. For example, rather than naming two workspaces *BUDGET* and *OLDBUDGET*, it's easier to find them later if you name them *BUDGET* and *BUDGETOLD*. And, of course, the same idea helps out with function and variable names.

2. Wil 3-Chr Nms Run Fst?

Perhaps you've heard the scam: "Hey buddy, wanna make your APL functions run **real fast**? ..jes' keep all of the names to three characters or less..."

Well, first of all, it **is** true that APL has to do more work to use a name that's over three characters long. **But**... that extra amount of work that it has to do is **trivial**. You may not even be able to measure the difference, whereas elsewhere in this paper we speak of other changes in programming technique that can result in improvements of hundreds of times.

If there is ever any chance that a longer name will be more readable or more descriptive, vote in favour of the more meaningful name.

The One-liner Syndrome

In APL, a line of code isn't restricted to having just one or a small number of primitives; it can be of **any** arbitrary length. Indeed, if one so desired, a championship chess program could be written in one line. However, it is appropriate to mention at this point that JUST BECAUSE SOMETHING **CAN** BE DONE DOESN'T MEAN THAT IT **SHOULD** BE DONE.

One of the biggest criticims that APL receives is that it is often obtuse, opaque, and just plain hard to follow. Unfortunately, that's often true. But the fault isn't with APL, it's with the "clever" [sic] programmer who shows how much he can pack into one line.

The One-liner Syndrome is an affliction that seems to hit every APLer at some time in their APL career... I'm not sure why. I certainly wasn't bypassed; everything that I wrote used to be packed into very few lines, all of which were all hundred-character-long twisted barbed-wire tangles of circles, slashes, and stars, looking like the wrath of Zeus bolting from the heavens, or perhaps like that crack in the living-room wall. The one thing that they **didn't** look like was readable code.

Over the years I slowly came to my senses, after noticing all of the obstacles that I was throwing into my own path. The most obvious problem is that such functions can't be understood without extensive study. This should be reason enough to stop using them, since the author most often has to maintain his own code. But there are other arguments against one-liners.

One-liners can't be meaningfully traced, using the "*T*∆fname" facility. Trace shows only the last (leftmost) result of each line... and when the entire function is on one line, that's not too meaningful.

Since a one-liner normally has to hold **many** temporary results in intermediate storage while it's parsing the rest of the line, one-liners are much more susceptible to *WS FULL*s than other (more reasonable) functions. Many is the time that I couldn't get past a *WS FULL*, until I finally bit the bullet and broke the offending line into several shorter lines.

This brings up the next problem: one-liners are just plain hard to edit. On most systems the maximum allowable line length that can be edited using the del-editor is 130 characters. True, it could be edited as a character matrix, but doing that may commit you to **always** doing that, so don't, unless it's to break it into shorter lines.

Everything that we've discussed so far has had to do with **programmer** efficiency, not **program** efficiency. One of the arguments that I have heard in **favour** of one-liners is that such a function will run faster. I always wonder if the proponents of this school of thought have actually tried any timing tests. I urge you to do so. One-liners aren't by nature faster **or** slower, but in actual practice, they are normally very much slower. One of the reasons for this is what I shall call "fake catenation" (a contrived phrase, but I think it communicates).

"Fake catenation" is an artifical construction that is employed as a trick simply to glue two lines together into one line. It should be avoided like the plague. For example, let's assume that we wish to assign a value of 0 to both *A* and *B*. One method would be to have two lines, *A*←0 and *B*←0. There is nothing wrong, though, with *A*←*B*←0. The computer doesn't have to do any extra work (and neither does the programmer when he reads the code six months later). However, let's now assume that we wish to assign *A*←0 and *B*←1. My preferred method would be two lines, stated just as they are in this statement of the problem. But a construction that I see all too often is *B*←1+*A*←0. Notice that this time APL has to do an extra addition that it wouldn't otherwise have to do; the addition isn't part of any of the productive work that the function is trying to accomplish. Now of course, the time to do an addition is extremely small ... but then so is the time to go to a new line.

A larger problem associated with the above example is in readability. Sure, that last example isn't **too** bad, but let's suppose now that you were going to set a *A*←19 and *B*←457. If that's stated in one line (and I often see examples like this), you have your choice of *B*←438+*A*←19 or *A*←¯438+*B*←457. Now, try to read **that** and know what the values of the two variables are ... to do so you have to mentally perform those same meaningless calculations that shouldn't be there in the first place. In an effort to prevent this problem, many programmers turn to the next (and more devastating) trick.

Using that last example, where we want *A*←19 and *B*←457, many programmers just automatically enter either *A*←19,0/*B*←457 or *A*←19,0ρ*B*←457. In each case, the value is assigned to *B*, then that value is reshaped in a null vector, catenated onto another value (19), and assigned to *A*. This is the real essence of fake catenation. One would expect that (other than the computer doing some extra work) this would be pretty much the same as entering those two statements on two lines; in particular, the resultant values in *A* and *B* would still end up being the same. ...'taint so, McGee.

Several times I have gotten calls from people who claim to have mysterious APL errors that just seemed to spring up out of the wood-work, when "Not Much Of Anything" had been changed in the code. Well, picture the following two cases:

```
       ∇FN1[⎕]∇
     ∇ FN1 M;C;H;L                  ∇FN2[⎕]∇
[1]    C←0                        ∇ FN2 M;C;H;L
[2]    H←1↑ρM               [1]     C←0,0ρH←1↑ρM          -set up variables
[3]   LP:C←C+1              [2]    LP:C←C+1               -increment counter
[4]    →(C>H)/0             [3]     →(C>H)/0              -check for the end
[5]    L←M[C;]              [4]     L←M[C;]               -get the next line
[6]    'THIS IS ',L         [5]     'THIS IS ',L          -print the line
[7]    →LP                  [6]     →LP                   -go back to loop
     ∇                            ∇
```

Well, the differences seem innocuous enough. And except for a slight case of fake catenation in *FN*2, the two functions appear to be identical. Oh yes, there is one other difference: *FN*2 doesn't work.

```
       FN2 MATRIX
LENGTH ERROR
FN2[5] 'THIS IS ',L
                   ∧
```

This problem is due entirely to the fake catenation, but that may not be at all apparent to the person who has to fix it at this point.

In *FN*1 with *C* assigned on a separate line, *C* took on a **scalar** (dimensionless) value of 0. In *FN*2, ",0ρ" caused the value that was put into *C* to be a **one-element vector**. Throughout the operation of these functions, *C* retains its given dimension. On the line containing *L*←*M*[*C*;], *C* is used as a subscript for a matrix. The shape of the result of any subscripting is the same as the catenation of the shapes of each of the subscripts [i.e. (ρsub1),ρsub2]. No column positions are indicated, so all columns are selected. In *FN*1, the scalar subscript causes *L* to be a vector, while in *FN*2, the vector subscript causes *L* to be a 1×n matrix.

Now, the statement that's trying to print the words "*THIS IS*" in front of the line has no problem when *L* is a vector. But when *FN*2 attempts to catenate an eight-element vector ("*THIS IS*") with a 1×n-matrix, a *LENGTH ERROR* results. At this point, there isn't likely to be any suspicion directed toward *C* until much time has been wasted.

This is the real problem with fake catenation: unless great care is exercised, it often produces treacherous (and elusive) side effects.

In general, one-liners are a waste of time... yours and the computer's. Abolishing one-liners from your library can save you needless errors and debugging time, and can ease your lot in life in understanding the code. It can also save you *WS FULL* problems and let your code run faster... saving you money and more of your own time.

Much more productivity can be realized if each line of an APL function contains **one thought**.

Summary

We certainly don't purport to have all of the answers... nobody does, don't kid yourself. The things that we've presented in this paper aren't necessarily **the** proper way of doing things.... they are simply some things that have worked for us. By no means are they intended to be Programming Standards. Such standards are often relatively meaningless outside of a single group working on a shared project. But over the years, we have gone through all of the normal mistakes and inefficiencies that everyone does, and undoubtedly still have many more to struggle our way through. The hints that we have included in this paper are all things that have saved us considerable time and grief. We expect that they can do the same for you.

If you are an old-timer to APL, passing these hints along to some of the beginning APLers in your department may enable you to benefit from their increased productivity.

MABRA: A PACKAGE FOR RECORD HANDLING SYSTEMS

Hugh Hyndman
I.P. Sharp Associates Limited
Toronto, Canada

MABRA is a package that provides a generalized record administration facility. This means that any system that can be conceptualized as a collection of fixed records of data, can be easily managed by MABRA.

What is a record of data? Well, take an employee within a personnel system, a stock item within a warehouse system, or a product within a manufacturing system. Each of these items can be considered a **record** of data.

Each of these records has certain attributes: each employee in a personnel system has a name, salary, grade, and department; each product in a manufacturing system may have a model number, serial number, unit cost, and retail destination. These attributes are referred to as **fields** in MABRA.

The whole idea of managing a number of records is an old concept in data processing. Software to maintain and analyze these records has been written by APL users several times over. MABRA was developed in an attempt to stop the duplication of effort and to help users solve their problems **quickly** and **easily** (often with no knowledge of APL).

Figure 1 illustrates a typical group of records that can be managed by MABRA. The value of each field is shown. Each row relates to a particular record. Each column contains a particular field. At the top of the diagram, outside the box, the names of the fields are written. Each field has its own data type (e.g. character, integer, real, binary, or date) and its own fixed length (e.g. 15 characters, 2 integers).

MABRA was developed in the U.K., when I had some ideas about a record management system, but no application. By chance, B.L. Cars (British Leyland) was searching for a system to maintain records for their 100,000 employees. This system was to be flexible, easy to use, and able to quickly cope with organizational changes. So, with the much appreciated assistance of Chris Baron (B.L. Cars), MABRA was developed.

Initial Design Considerations

During the initial stages of developing MABRA, the following design goals were considered:

1. General Applicability

How general should I make it? If it's too general, then user costs will soar and it will become too difficult to use. If it's not general enough, then it won't be useful. In order to balance these two conflicting goals, I settled on the following minimum requirements:

a) ability to add, modify, or delete fields from the system at any point in time.
b) ability to search on any field.
c) ability to add, modify, or delete records.
d) ability to display records.

Once these requirements were met, I also implemented additional features, all independent of the basic needs and therefore not affecting the running costs of those who didn't use the new features. These additional features were developed to handle multi-user access, highspeed printing of records, and crosstabulation analysis.

2. Simplicity

It is very easy after any software project has been completed, to put in "bells and whistles", or modify the software "slightly" to handle "just another" special case. However, all these enhancements cut into processing costs, and on occasion they make the original easy-to-use system difficult and clumsy. Therefore, I tried to keep MABRA fairly basic, and offered the user a utility workspace (MABRAUTIL), so that special reports and updates could be written outside the actual package.

Directly related to keeping the system simple, is the necessity of keeping the user interface simple. Obviously, users shouldn't have to worry about the details of the file structure, and how things work internally. It is difficult enough for first-time users to learn how to sign on without the additional confusion of new computer terminology such as *⎕TS*, *⎕READ*, hashing, etc. MABRA is prompt-driven by English-language commands. If the user is confused at any point, all he has to do is type *HELP*. MABRA will print a one-line message to remind him what to do. If that isn't enough, he can type *HELP* again, and more detailed assistance will be printed. The *HELP* messages describe all the options available, and if the user accidentally enters an invalid response to a prompt, MABRA will just reprompt for a new response after explaining what is unacceptable with the original response.

3. Speed of Set-up

Now that we have a simple and somewhat general package, how long does it take to set up a system? Does the user know what fields he needs in the system? If he does, then it's as quick as entering the names, types, and widths of each field. From there, he has to input the records.

Also, if extra programming is required for special updates and reports, then the utility functions in MABRAUTIL relieve the programmer of the necessity of digging into the detailed structure of the system. Therefore, even special purpose programs can be implemented quickly.

Implementation Problems and Solutions

With the design rules laid down before me, it should have been clear sailing from there on in. Well, it didn't turn out that way. I ran into some tough implementation problems and in this section, I would like to discuss some of them and how I solved them. Hopefully, you can use some of these ideas when developing other systems.

1. Flexibility & Simplicity of Data Management

In many applications, it is necessary to change the specifications of a system during or after development. This may be due to re-evaluating the system's objectives, encompassing another part of the organization, or spawning new applications off the system on account of the system's past successes. For example, a successful personnel system might have pension information or company car information appended to it.

These changes of specification could necessitate the addition of new fields, the deletion of existing fields, or the addition of more records, consequently requiring a system that is open-ended (both in fields and records) as well as one that would make it possible for users to simply make these changes.

All the records for a particular MABRA system are stored in a **master file**. Each component of this file contains a **fixed** number of records (one could say that each component is blocked). Also, each of these components contains a package of variables which represent each of the fields in the system. Each variable (field) is a matrix, with each row across all the variables in the package holding data for each record.

Figure 2 illustrates a typical MABRA master file. This file holds 22 records and is blocked at 20 records per component. Note that the last component of the file is blocked at 20 records even though only 2 records are held in it. This blocking method is used to allow for more records to be added to the system minimizing the problem of "growing" file replaces. Each record consists of three fields: *NAME*, *SEX*, and *SALARY*. The data for these fields are stored in variables *⍙NAME*, *⍙SEX*, and *⍙SALARY*. The delta-underscore (⍙) prefix is used to prevent name conflicts with the MABRA software when these variables are read into the workspace.

Each record in the master file has its own record number starting from 1 to the current number of records in the system (in this case the current number of records is 22). Given a list of record numbers, it is trivial to calculate the location of each record in the master file.

```
COMPONENTS←⌈RECORDNUMBERS÷BLOCKINGFACTOR
ROWS←1+BLOCKINGFACTOR|RECORDNUMBERS-1
```

The following user input into MABRA adds a field called *LOCATION* to the system illustrated in figure 2. Note that the user input always appears after a colon.

```
* ADD FIELD
* ENTER FIELD NAME: LOCATION
* TYPE? (CHAR/INTE/REAL/BINA/DATE): CHAR
* NUMBER OF ELEMENTS IN FIELD: 3
* USE DEFAULT OUTPUT FORMAT? (Y/N): Y
* OK TO ADD? (Y/N): Y
* FIELD ADDED
```

Given this input, MABRA creates a new variable

```
⍙LOCATION←20 3 ⍴' '  ⍝ DEFAULT VALUE OF BLANKS
```

and then inserts (*⎕PINS*) this variable into every component of the master file. Since file components are expansible, there is no problem with increasing the record size. Adding a field also requires updating directories containing information about that field's name, type, width, output format, etc. In view of the fact that the addition of fields is usually done when the system is first being defined, the "growing" replace problem is insignificant. Once there are a large number of records in the system however, this problem will usually increase the file storage until the scheduled full file backup is performed.

The following input into MABRA deletes the field *SEX* from the same system:

```
* DELETE FIELD
* FIELDS TO BE DELETED: SEX
* DELETE SEX? (Y/N): Y
* DELETED.
```

Deleting the *SEX* field requires expunging (⎕*PEX*) the variable ⍙*SEX* from each component of the master file, as well as removing entries from the directories mentioned above. Changing a field's width simply requires reading every file component of the master file modifying the appropriate variable.

Adding a new record to the system is done by prompting for values of all the fields making up a record.

```
* ADD RECORD
NAME     :PATTISON
SALARY   :23000
LOCATION :TORONTO

NAME     :STOP
* UPDATE REQUEST NO. 4 FILED.
```

The above input to MABRA makes a **request** to add a new record. This request is processed when the update task is started (through the *UP* command). MABRA takes the field value input and inserts these values into the appropriate variables in the master file. If all the available record space has been used up in the last file component, then another component (block) is appended to the master file, and the record is inserted there.

2. Searching Records

Once there are records in a system, you may wish to retrieve them in order to display, delete, or make changes to them. To identify these records, you should be able to specify an expression that is simple, but powerful enough so that an intricate search (one that may require combinations of different fields) may be done.

MABRA has such an expression, called a **constraint**. Basically, a constraint has the following format:

<field name> <relation> <value>,

where <field name> represents the name of the field that is to be searched, <value> is the data to be searched for, and <relation> is how the data is to be searched. For example,

```
PARTNO=ABC357
```

would find all parts (in a parts tracking system) whose part number field was equal to *ABC*357.

The "and" and "or" connectives, as well as parentheses, may be used if a more complicated search is needed to find certain records. For example,

```
(SEX=M∧AGE>60)∨(SEX=F∧AGE>55)
```

would find all people in a personnel system who were males over 60 years of age **or** females over 55 years of age. You will observe that the above constraint is not an APL expression. The order of execution is left to right, with the relations having precedence over the connectives. The reason I decided on this "conventional" order of execution is that those who do not have APL experience would find this method much easier to learn and read.

MABRA allows you to search on **any** field. However, if certain fields are frequently used as search fields, then those fields should be organized in such a manner so that MABRA uses the least amount of computer resources to complete a search. The method MABRA uses is to invert the field. Inverting a field means taking a copy of all the values for a particular field, and storing them together in a secondary file (inverted file). Storing all the field values together reduces the number of file operations for searching. Once more, only fields that are frequently used as search fields should be inverted, because inverting a field duplicates the storage of that field, as well as requires updates to be performed on two files (master file and inverted file).

Because of the generalities of field names, widths, and values, as well as the use of complicated search constraints, MABRA generates its own APL functions to process the search. In the original implementation, as MABRA checked for validity of the search expression, it did the actual search. Therefore, when the validity check was completed, so was the search. This method proved the most efficient if the search expression was valid. However, if the search expression was invalid, then a considerable amount of computer resources would be consumed to ascertain this invalidity. Now, MABRA does a validation pass first, which generates APL functions to do the search; if the expression is valid, these APL functions are called.

3. Displaying Records

Now that we have a method of selecting certain records, how are they to be displayed? Are only certain fields within the records to be printed? Are the records to be sorted in a more readable fashion? How are the field values to be formatted as output?

The solution is quite simple. Initially, use the search mechanism discussed in the previous section to identify which records are to be displayed. Secondly, prompt the user for which fields are to be printed and which fields are to be used for sorting. From here, all that has to be done is to read the fields of the records from the master file, sort them, and then by using each of the print field's format phrases (left argument of ⎕*FMT*), print them. By knowing the printing width of each field, column headings are simple to apply.

The following input into MABRA demonstrates the ease of displaying records:

```
01:LR
* LIST RECORDS
* CONSTRAINTS: LOC=0501
    53 FOUND.
* PRINT FIELDS: SURNAME,INITS,DOBIRTH,JOBTITLE
* SORT FIELDS: SURNAME,INITS
* ALIGN PAPER, TYPE SPACE, CR TO START.

SURNAME          INITS  DOBIRTH JOBTITLE

ATHERTON         RF    17/05/18 TRAINEE OPERATOR
AYRES            RCS   30/07/43 OPERATOR
BAKER            A     16/07/23 PROGRAMMER-ANALYST
BAKER            BJ     4/10/37 PROGRAMMING SUPERVISOR
BATEY            S     18/02/33 OPERATOR
BIGGS            GN    31/03/37 SENIOR PROGRAMMER
BOUCHER          MA     9/11/36 SENIOR PROGRAMMER
BRETT            D     19/01/22 OPERATOR
BRIGHT           HG    28/05/49 KEYPUNCH OPERATOR
BURTON           NA     6/11/48 . . .
```

Fifty-three records are identified with their *LOC* field equalling 0501. Using these records, the *SURNAME*, *INITS*, *DOBIRTH*, and *JOBTITLE* fields are printed in *INITS* within *SURNAME* ascending order.

The actual formatting of the records for output is done in a few lines line of APL! Once the selected records are in the workspace, MABRA builds up both arguments of ⎕*FMT* (format list and data list), and then calls ⎕*FMT*.

For example, suppose three fields with the following names and format phrases are to be printed:

```
NAMES←→NAME
       DOBIRTH
       SALARY

FORMATS←→15AI
         G⎕ZZ/ZZ/Z9⎕
         CI6
```

and the data for each field is stored in variables whose names are the names of the fields prefixed by a delta (Δ). The APL required to output those fields using their correct format would be:

```
FMT←¯4↓,FORMATS,3 4ρ',X2,'  ⍝ 2 spaces between each field
DATA←'(',(1↓,';','Δ',NAMES),')'
⍎'FMT ⎕FMT ',DATA
```

A few months after MABRA became a public package, it became necessary to implement coded fields. Coded fields are fields whose values are coded or abbreviated versions of their full descriptions. Only the codes are stored with the record; full descriptions may be printed out when listing records. A table of codes and their respective descriptions is defined by the user and is stored independently of the records. Using codes saves storage space in the master file, as well as reduces the amount of data input. Also, if codes are well designed, they can be very useful in record searching (grouping of records, etc.).

The following user input illustrates how codes are used in searching and how the code descriptions are displayed:

```
01:LR
* LIST RECORDS
* CONSTRAINTS: LOC=05
   145 FOUND.
* PRINT FIELDS: SURNAME,SALARY,LOC,(LOC)
* SORT FIELDS: -SALARY
* ALIGN PAPER, TYPE SPACE, CR TO START.

SURNAME         SALARY LOC  (LOC)

OBRIEN          17,464 0501 DUBLIN, EIRE
NEVE            10,342 0504 LIMERICK, EIRE
JENSEN           9,651 0501 DUBLIN, EIRE
STANLEY          9,439 0504 LIMERICK, EIRE
TROW             9,391 0505 GALWAY, EIRE
FARR             9,112 0503 WATERFORD, EIRE
FLYNN            8,971 0505 GALWAY, EIRE
DAVIES           8,715 0502 CORK, EIRE
WEBSTER          8,647 0504 LIMERICK, EIRE
MITTELL          8,584 0502 CORK, EIRE
TRINDER          8,556 0503 WATERFORD, EIRE
BATCHELOR        8,296 0501 DUB . . .
```

The *LOC* field is a four character field, containing a location code. The first two characters indicate which country the employee is located in (in this case 05 means *EIRE*), and the whole field indicates which city the employee is located in. The search for match is performed on the first two characters of *LOC*, therefore all location codes with their first two characters equal to 05 would be identified. Within these identified records, the *SURNAME*, *SALARY*, *LOC*, and description of *LOC* fields are printed in *SALARY* descending order.

If too many records are selected for sorting and printing at the terminal, the workspace size may not be large enough to sort all the records. This problem is solved by using the *BP* (batch print) command. The *BP* command uses a batch file sort utility, so all the records to be printed are put in a sort input file, sorted, and then formatted for output. The output can be either directed to the highspeed printer or back to the terminal (via another APL file).

4. Modular System Design

How can we design the system so that only one copy of the MABRA software is shared by all users? This design should also take into consideration modifying the software so that all users using MABRA can benefit from these changes. Also, with all the functions used by MABRA (over 300K bytes of them), how are they stored and run?

All functions and variables used by MABRA are stored in packages on file. The software for each command (both in system mode and data mode) is stored in a different component. Each such component is called a module.

Figure 3 illustrates the basic modules in MABRA. When a command is invoked, MABRA reads the module into the workspace. When that command is finished, all the functions and variables in that module are expunged.

For example, the *AR* (add record) command executes a function similar to the following:

```
     ∇ ADDRECORD;DOADDRECORD;BUILDXEQS;GETIND;REQUPD
[1]    ⍝ INVOKES 'AR' MODULE
[2]    ⎕PDEF ⎕READ FUNCTIONFILE,ARCOMP ⍝ READ IN MODULE
[3]    DOADDRECORD ⍝ MAIN 'AR' DRIVER FUNCTION
```

This function reads in the functions: *DOADDRECORD*, *BUILDXEQS*, *GETIND*, and *REQUPD*. These functions are **local** to *ADDRECORD*, and therefore are expunged automatically on exit from *ADDRECORD*. The function *DOADDRECORD* is the main driver function in the *AR* module.

The MABRA workspace has only one function in it, *START*, which starts up the MABRA software. This is the only function that a user can store in his private workspace by doing a *)COPY*. It is thus designed so as to contain the minimum amount of software and will never need modification.

```
     ∇ START;FUNCTIONFILE;GO
[1]   ⍝ START UP MABRA
[2]    ⎕UNTIE ⎕NUMS  ⍝ UNTIE ALL FILES
[3]    '86 MABRA' ⎕STIE FUNCTIONFILE←1 ⍝ TIE SOFTWARE FILE
[4]    ⎕PDEF ⎕READ FUNCTIONFILE,1
[5]    GO
[6]    ⎕UNTIE FUNCTIONFILE
     ∇
```

All this function does is open (tie) the MABRA software file, read in the main startup function (*GO*), and then execute *GO*. Since the function *GO* is stored on file, all the functions and variables that start up the MABRA software can be localized within *GO*.

There are two versions of the MABRA software: the production version and the test version. The test version is used for developing new enhancements to the software. Once an enhancement has been suitably debugged in the test version, it is installed into the production version. Both copies of the software (before and after enhancement) are kept in the production version as a safeguard until the enhanced software has been proven. A workspace has been written to easily update both the production and test versions of the MABRA software. Since all MABRA commands are stored in separate file components (modules), it is possible that a module can be updated at the same time users are using MABRA, so that when a user invokes that module, the updated software will be used.

Through the use of event trapping, the occasional software bug, as well as user inflicted errors are reported immediately to the MABRA support team (in Toronto, London, Zurich, and Rochester) via Sharp's electronic mailbox (666 *BOX*). This is done so that errors may be dealt with on a "round the clock" basis, and so that users may be notified of what went wrong. Here is typical mailbox message giving details of an error:

```
NO. 1101400 FILED 17.56.44  MON 28 JUL 1980
FROM MDEV
TO   HHY RAJ AWE SJHAL

USER=1234567 USER;BCCC=10101 1234001;NAME=MABRA USER
FILE RESERVATION ERROR
ΔRSZ[4] T[1] ⎕RESIZE A
        ∧
FILES=86 MABRAV2;1234567 ACC60,DIR60,INV60,MAS60,MSK60,SYS,UPD60,S̲728175639
)SI=ΔRSZ[4]⋆;XEQ2[1];REBLOCK[38];REBLOCK[3];⍎;ENTERSYSTEM[22]
;ENTERSYSTEM[2];⍎;SYSTEM[9];GO[21];START[13];START[5]
```

This message indicates that the user 1234567 (in Toronto) ran into file reservation problems while using the *RB* (master file reblock) command in his system 60. The error occurred at 17:56 on July 28.

Every user's MABRA system has a software version number stored with it. This is done so that if a **new** version of MABRA is to be released, that may require a small modification to the user's files, then it can be done automatically by MABRA. The MABRA software checks if its version number differs from the version number of the user's MABRA files, and if so, then special conversion functions are called to upgrade the user's files to the new software. This conversion is done when the user enters data mode.

5. Multi-user Systems

In many companies, it is necessary for several people to access the same system. Take, for example, a personnel system for a multi-location company. It may be required that each location be able to update and report on its own personnel, as well as be able to report on selected information on a company-wide basis. Head office could have access to all the data, whereas each location would be limited to access to its own data.

By default, the owner of the MABRA system is the **only** user who has access to it. If desired, the owner may grant access to his system to other users. There are three different types of access that may be granted:

1) command access - allows a user to invoke certain commands
2) record access - allows a user access to certain records
3) field access - allows a user access to certain fields (for displaying, adding, or changing)

When a user enters a MABRA system to which he has only partial access, the part to which he doesn't have access is invisible to him: *HELP* won't refer to those parts, and *ALL* records or *ALL* fields will refer to only the records and fields he can access. Error messages will not inform him of an access error but just state that it's an incorrect command.

The update commands (*AR*, *CR*, *CD*, and *DR*) do not actually modify a MABRA system, but rather **request** that the system be modified. MABRA has a separate command for processing these requests (*UP* - initiate update task). This command initiates a non-terminal or batch task (N-task or B-task) to process all such pending requests. Requesting a modification is done for several reasons. Processing these requests in batch mode reduces system costs as well as simplifies restarts in case of system crashes. By allowing a separate procedure to perform updates, MABRA lets the user withdraw an update request, if he realizes he has made a mistake, **before** the update is processed. Moreover, if an update accidentally wipes out a major portion of the data, it is easy to have files retrieved from the previous file backup and then re-run the update using the requests (except the "villains") made from the time of the backup, to the present. Update requests are stored on file until the user is quite sure that they are no longer needed. At that point he may run a command to remove them.

A common question that arises in multi-user systems is, "When are critical commands run?". Critical commands are those that change the system structure, for example, adding, changing, and deleting fields. To solve this, MABRA has a "hard hold" mechanism. That is, a method which blocks other users until the hold is removed. A system can only be "hard held" if no other users are using the system **and** no update requests are pending. To tell if no other users are using the system, a log is kept of entries and exits. To ensure that this log is correct, an attempt to **exclusively** tie the master file is made. If that fails, then someone else is using the system:

```
      ⎕TRAP←'∇24 E →STILLINUSE'
      'MAS01' ⎕TIE MASTIE
```

The hard hold is achieved by the MABRA software turning an indicator on in one of the MABRA system files. If other users attempt to enter the system being held, a message will be printed indicating that the system is temporarily unavailable.

6. Related Fields

In some record administration systems, certain fields relate to other fields. For example, a field *PROFIT*, may relate to the fields *SALES* and *EXPENSES* where

```
      PROFIT←SALES-EXPENSES.
```

Rather than entering a value for *PROFIT* every time *SALES* or *EXPENSES* is modified, you can tell MABRA to update a field automatically, by giving it the necessary information about how those fields are related.

Another type of related field is one whose value is dependent on the fact that another field is being updated, and not necessarily what value is being assigned to it. For example,

```
      UPDATEDAY←100⊥100|3↑⎕TS    ⍝ DAY OF UPDATE
      UPDATETIME←100⊥3↑3↓⎕TS  ⍝ TIME OF UPDATE
```

would be assigned if any other field had been changed. Related fields can also be used for historical shifting, as well as shifting information from one MABRA system to another.

The following MABRA session shows how the above relations work.

```
05:LF
* LIST FIELD
* ALIGN PAPER, TYPE SPACE, CR TO START.
```

```
               05: RELATED FIELDS EXAMPLE      DATE:  7/08/80

NAME               TYPE WIDTH  INVERTED  ACCESS DESC-WID  ⎕FMT

OFFICE             INT      1            ACL              ZI2
SALES              INT      1            ACL              CI7
EXPENSES           INT      1            ACL              CI7
PROFIT             INT      1            ACL              CI7
UPDATEDAY          DATE     1            ACL              G⎕ZZ/ZZ/Z9⎕
UPDATETIME         INT      1            ACL              G⎕99:99:99⎕
```

Above, is a listing of all the fields in the sample MABRA system.

```
05:AR
* ADD RECORD(S)
* FIELD ORDER: 2

OFFICE      :1
SALES       :13405
EXPENSES    :3244

OFFICE      :2
SALES       :12000
EXPENSES    :16000

OFFICE      :3
SALES       :14000
EXPENSES    :200

OFFICE      :99
SALES       :145000
EXPENSES    :25600

OFFICE      :STOP
* UPDATE REQUEST NO. 5 FILED.
```

Here we are adding 4 records, but only inputting values for fields *OFFICE*, *SALES*, and *EXPENSES*. The other three fields are automatically updated. Here is a listing of the records after the update:

```
05:LR
* LIST RECORDS
* CONSTRAINTS: ALL
     4 FOUND.
* PRINT FIELDS: ALL
* SORT FIELDS: OFFICE
* ALIGN PAPER, TYPE SPACE, CR TO START.

OFFICE    SALES EXPENSES    PROFIT UPDATEDAY UPDATETIME

    01  13,405     3,244    10,161    7/08/80    15:31:04
    02  12,000    16,000    ¯4,000    7/08/80    15:31:04
    03  14,000       200    13,800    7/08/80    15:31:04
    99 145,000    25,600   119,400    7/08/80    15:31:04
```

Now we will change the *EXPENSES* field of office 02 to 1600:

```
05:CR
* CHANGE RECORDS
* CONSTRAINTS: OFFICE=2
     1 FOUND.
* PRINT? (Y/N): N
* ENTER CHANGES.
:EXPENSES←1600
:STOP
* LIST CHANGES? (Y/N): N
* FILE UPDATE REQUEST? (Y/N):Y
* UPDATE REQUEST NO. 6 FILED.
```

After running the update, here is the field values for office 02:

```
05:LR
* LIST RECORDS
* CONSTRAINTS: OFFICE=2
     1 FOUND.
* PRINT FIELDS: ALL
* SORT FIELDS:
* ALIGN PAPER, TYPE SPACE, CR TO START.

OFFICE   SALES EXPENSES  PROFIT UPDATEDAY UPDATETIME

    02  12,000    1,600  10,400   7/08/80   15:32:43
```

An Interesting Application

MABRA has been used for many applications since the original system at B.L. Cars. Not surprisingly, it has been used for several personnel systems. It has also been used for oil rig location tracking, equipment inventory, P.G.A. volunteer tracking, mailing lists, and the Boat People sponsoring system. One rather interesting application of MABRA is the Nominal Catch Data Base, a data base containing ICNAF statistics. ICNAF stands for "International Commission for North-West Atlantic Fisheries". In 1979, ICNAF was replaced by the North-West Atlantic Fisheries Organization (NAFO). These statistics contain information on the number of metric tons of fish caught by year, country, ICNAF division, and fish species. This is maintained by the Department of Fisheries and Oceans (Government of Canada).

This MABRA system contains 22,000 records, each of which contains the following fields:

Name of Field	Description
YEAR	year
CCOUNTRY	country code
ACOUNTRY	abbreviated country name
CICNAF	ICNAF division code
AICNAF	abbreviated ICNAF division name
SPECIES	species code
CATCH	nominal catch (in metric tons)

Here are some sample countries in the system:

CCOUNTRY	**ACOUNTRY**	**Description**
01	*BUL*	Bulgaria
02	*CANMQ*	Canada (Maritimes & Quebec)
03	*CANN*	Canada (Newfoundland)
04	*CUBA*	Cuba
09	*FRASP*	France (St. Pierre et Miquelon)
10	*FRG*	Federal Republic of Germany
15	*NOR*	Norway
21	*UK*	United Kingdon
22	*USA*	U.S.A.
20	*USSR*	U.S.S.R.
24	*IRE*	Ireland

and here are some sample species codes and names:

SPECIES	**Description**
101	Atlantic cod
102	Haddock
103	Atlantic redfish
148	Greenland cod
278	Big eye tuna
539	Scallops
622	Northern Lobster
672	Horseshoe crab
469	Sharks

Using the *LR* (list records) command to obtain a list of species names for the Canadian catch in Division 3M in 1976:

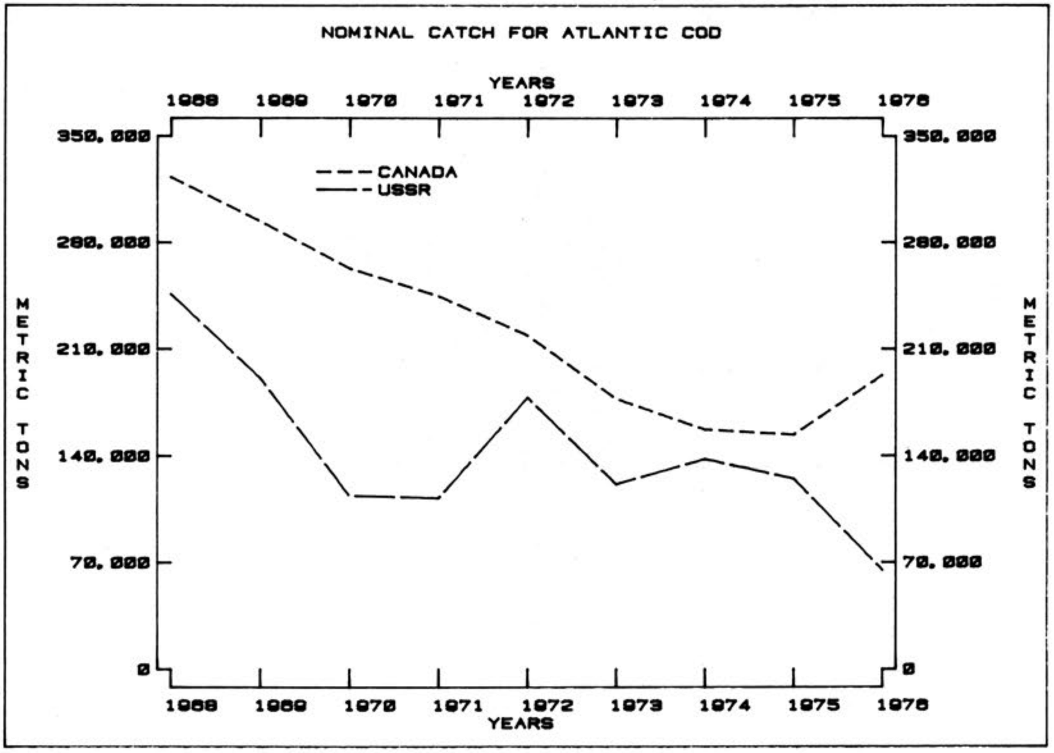
```
10:LR
* LIST RECORDS
* CONSTRAINTS: YEAR=76∧AICNAF=3M∧ACOUNTRY=CAN
   16 FOUND.
* PRINT FIELDS: SPECIES,(SPECIES),CCOUNTRY,(ACOUNTRY),CATCH
* SORT FIELDS: CCOUNTRY,SPECIES
* ALIGN PAPER, TYPE SPACE, CR TO START.

SPECIES (SPECIES)             CCOUNTRY (ACOUNTRY)                       CATCH

    101 ATLANTIC COD                02 CANADA (MARITIMES ∊ QUEBEC)         80
    103 ATLANTIC REDFISH            02 CANADA (MARITIMES ∊ QUEBEC)       4040
    112 AMERICAN PLAICE             02 CANADA (MARITIMES ∊ QUEBEC)          2
    114 WITCH FLOUNDER              02 CANADA (MARITIMES ∊ QUEBEC)          1
    116 YELLOWTAIL FLOUNDER         02 CANADA (MARITIMES ∊ QUEBEC)          1
    120 ATLANTIC HALIBUT            02 CANADA (MARITIMES ∊ QUEBEC)          1
    186 WHITE HAKE                  02 CANADA (MARITIMES ∊ QUEBEC)          2
    188 WOLFFISHES (=CATFISHES)     02 CANADA (MARITIMES ∊ QUEBEC)          8
    199 GROUNDFISH (NS)             02 CANADA (MARITIMES ∊ QUEBEC)         45
    101 ATLANTIC COD                03 CANADA (NEWFOUNDLAND)               54
    103 ATLANTIC REDFISH            03 CANADA (NEWFOUNDLAND)             4328
    112 AMERICAN PLAICE             03 CANADA (NEWFOUNDLAND)              189
    116 YELLOWTAIL FLOUNDER         03 CANADA (NEWFOUNDLAND)                2
    118 GREENLAND HALIBUT           03 CANADA (NEWFOUNDLAND)                1
    120 ATLANTIC HALIBUT            03 CANADA (NEWFOUNDLAND)                1
    188 WOLFFISHES (=CATFISHES)     03 CANADA (NEWFOUNDLAND)               17
```

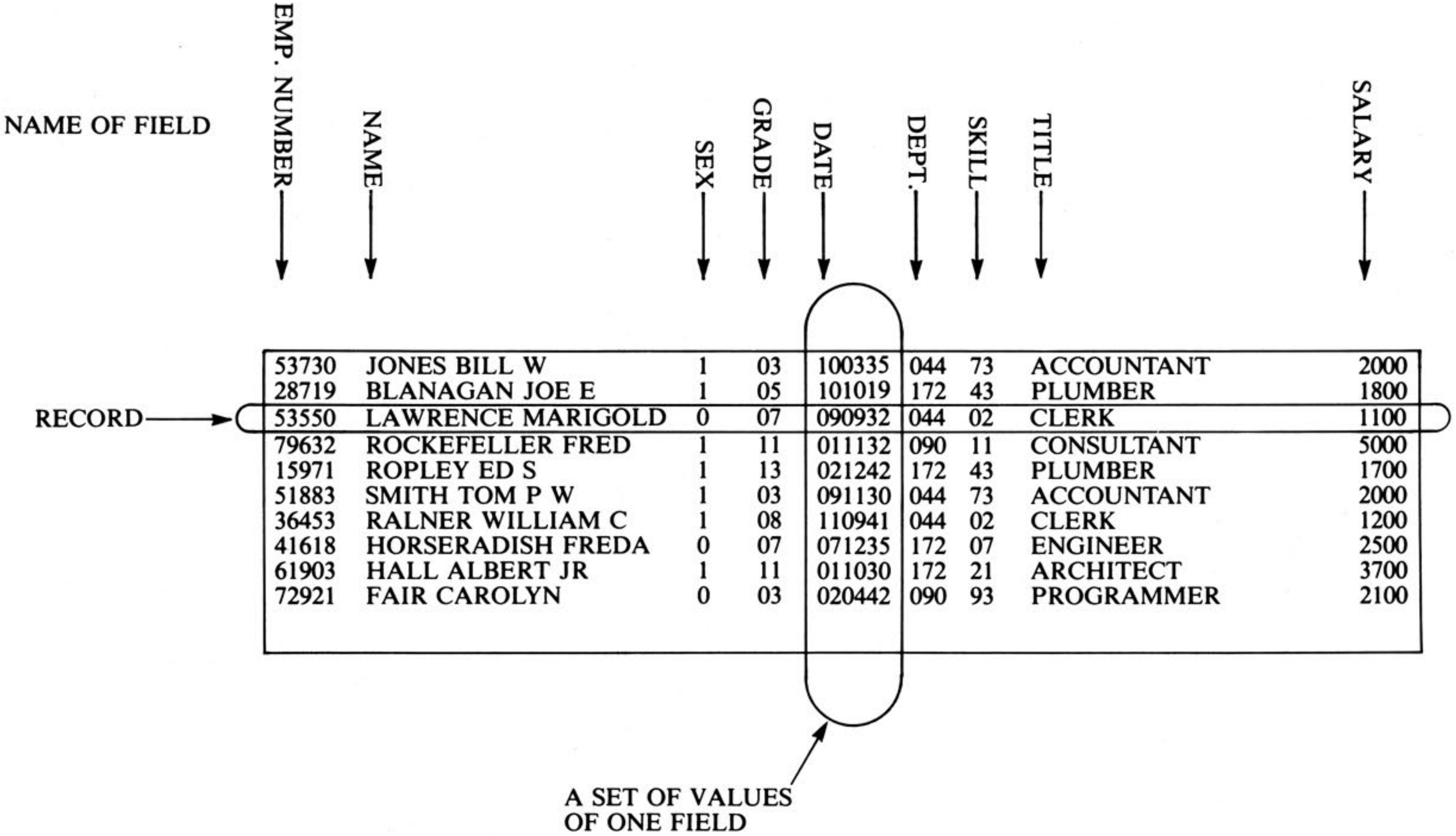
Using the *AN* (crosstabulation analysis) command to find the nominal catch of Atlantic cod, Haddock and Redfish for Canada - Maritimes and Quebec, by year (65-76) and by species:

```
10:AN
* CROSSTABULATION ANALYSIS
* OPTION? (F/T/A/FT/FA/FTA/TA): T
* CONSTRAINTS: ACOUNTRY=CANMQ∧(SP=101∨SP=102∨SP=103)
  492 FOUND.
* FIELD NAME LIST: YEAR,SPECIES
* USE RANGES? (Y/N): N
* FIELD TO BE TOTALLED: CATCH
* DO ANALYSIS? (Y/N): Y
* ALIGN PAPER, TYPE SPACE, CR TO START.

                         10: NOMINAL CATCH DATA
* TOTALS. CONSTRAINTS: ACOUNTRY=CANMQ∧(SP=101∨SP=102∨SP=103),
  ATTRIBUTES: YEAR,SPECIES, TOTALLING  CATCH

                         C1       C2       C3      TOTAL
               +----------------------------------
YEAR=        65|  123958    46993    28687     199638
    =        66|  120475    58980    47769     227224
    =        67|  109588    54144    50518     214250
    =        68|  121819    48341    61005     231165
    =        69|  114678    41770    64008     220456
    =        70|  110127    24927    65307     200361
    =        71|  105194    27768    83714     216676
    =        72|  102477    16326    78763     197566
    =        73|   80169    17701   108390     206260
    =        74|   73610    14388    62228     150226
    =        75|   76252    19124    62285     157661
    =        76|   72914    19294    49864     142072
          TOTAL| 1211261   389756   762538    2363555


 C1 : SPECIES=          101
 C2 :        =          102
 C3 :        =          103
```

Now, using MABRAUTIL and SUPERPLOT, a plot of the nominal catch for Atlantic cod for the USSR and Canada by year (1968-76) can be produced.

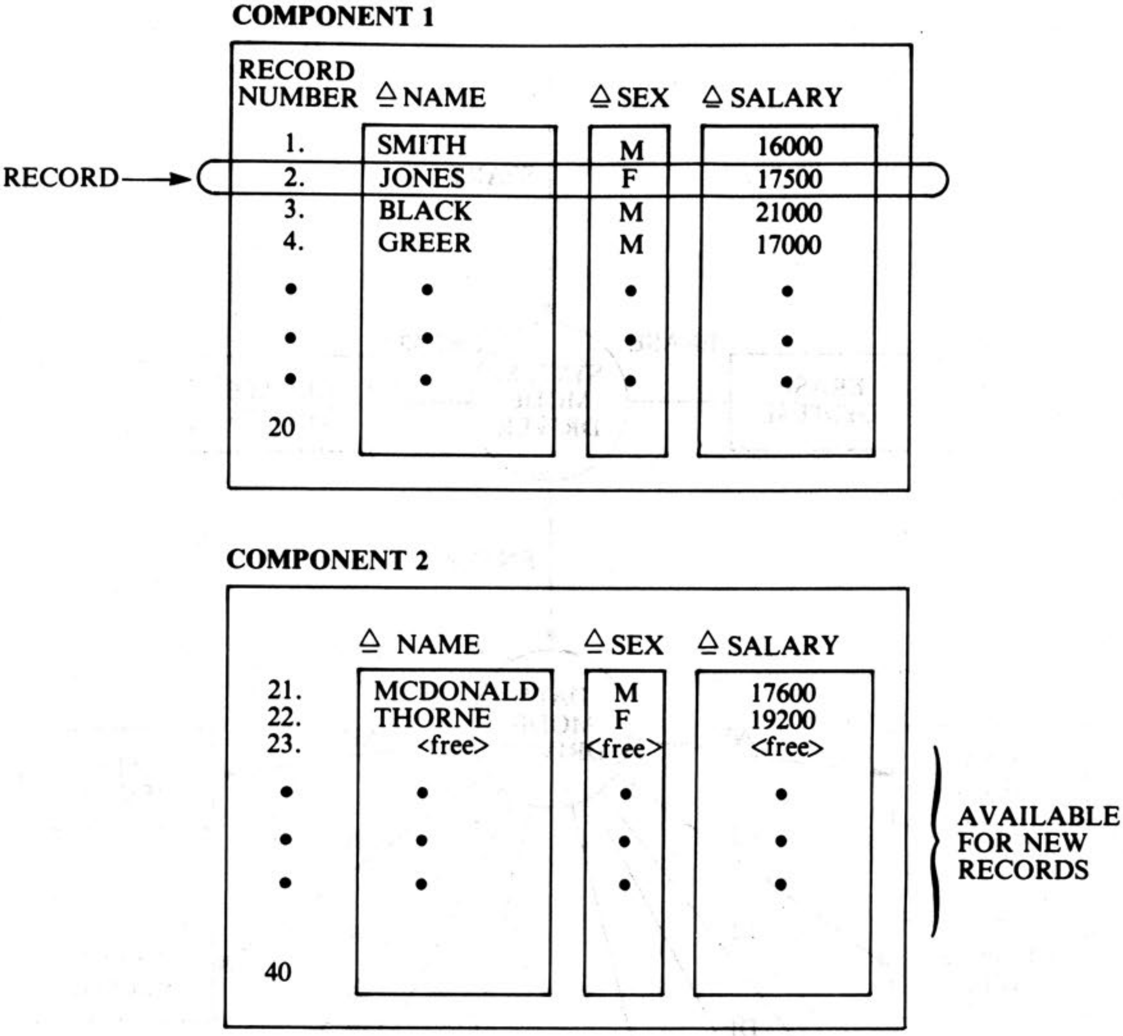
```
      )LOAD 85 MABRAUTIL
SAVED  20.07.36 08/06/80
COPYRIGHT (C) I.P. SHARP ASSOCIATES LIMITED 1978, 1979, 1980.
      ENTER 10
      SEARCHFOR 'CCOUNTRY∊02 03 20∧SPECIES=101∧YEAR≥68∧YEAR≤76'
0
      +/FOUNDMASK
411
      411 READ 'CCOUNTRY,YEAR,CATCH'
411
      X←(,ΔCCOUNTRY∊2 3),[.5] ,ΔCCOUNTRY=20
      Y←(67+⍳9)∘.=,ΔYEAR
      FREQ←X+.×⍉Y×(⍴Y)⍴ΔCATCH
      FREQ
322872 293857 262922 244611 218799 177283 157138 153746 192717
245956 190882 113599 111996 178083 121408 137878 124908  64995
      ⎕SP←FREQ
      )LOAD 3 SUPERPLOT
SAVED  18.46.48 07/15/80
      YEARLY, DATED 1968 TO 1976
      LABEL 'CANADA,USSR'
      TITLE 'NOMINAL CATCH FOR ATLANTIC COD'
      ΔSUPERPLOT 'TERM,HP/STYLE,LDASH,SDASH/COLOUR,BLACK,BLACK'
      ΔSUPERPLOT 'RANGE,0 350000/YLABEL,METRIC TONS'
      'HTBLR' PLOT ⎕SP
```





EMP. NUMBER	NAME	SEX	GRADE	DATE	DEPT.	SKILL	TITLE	SALARY
53730	JONES BILL W	1	03	100335	044	73	ACCOUNTANT	2000
28719	BLANAGAN JOE E	1	05	101019	172	43	PLUMBER	1800
53550	LAWRENCE MARIGOLD	0	07	090932	044	02	CLERK	1100
79632	ROCKEFELLER FRED	1	11	011132	090	11	CONSULTANT	5000
15971	ROPLEY ED S	1	13	021242	172	43	PLUMBER	1700
51883	SMITH TOM P W	1	03	091130	044	73	ACCOUNTANT	2000
36453	RALNER WILLIAM C	1	08	110941	044	02	CLERK	1200
41618	HORSERADISH FREDA	0	07	071235	172	07	ENGINEER	2500
61903	HALL ALBERT JR	1	11	011030	172	21	ARCHITECT	3700
72921	FAIR CAROLYN	0	03	020442	090	93	PROGRAMMER	2100

Figure 1



COMPONENT 1

RECORD NUMBER	≙ NAME	≙ SEX	≙ SALARY
1.	SMITH	M	16000
2.	JONES	F	17500
3.	BLACK	M	21000
4.	GREER	M	17000
•	•	•	•
•	•	•	•
•	•	•	•
20			

COMPONENT 2

	≙ NAME	≙ SEX	≙ SALARY
21.	MCDONALD	M	17600
22.	THORNE	F	19200
23.	<free>	<free>	<free>
•	•	•	•
•	•	•	•
•	•	•	•
40			

Figure 2

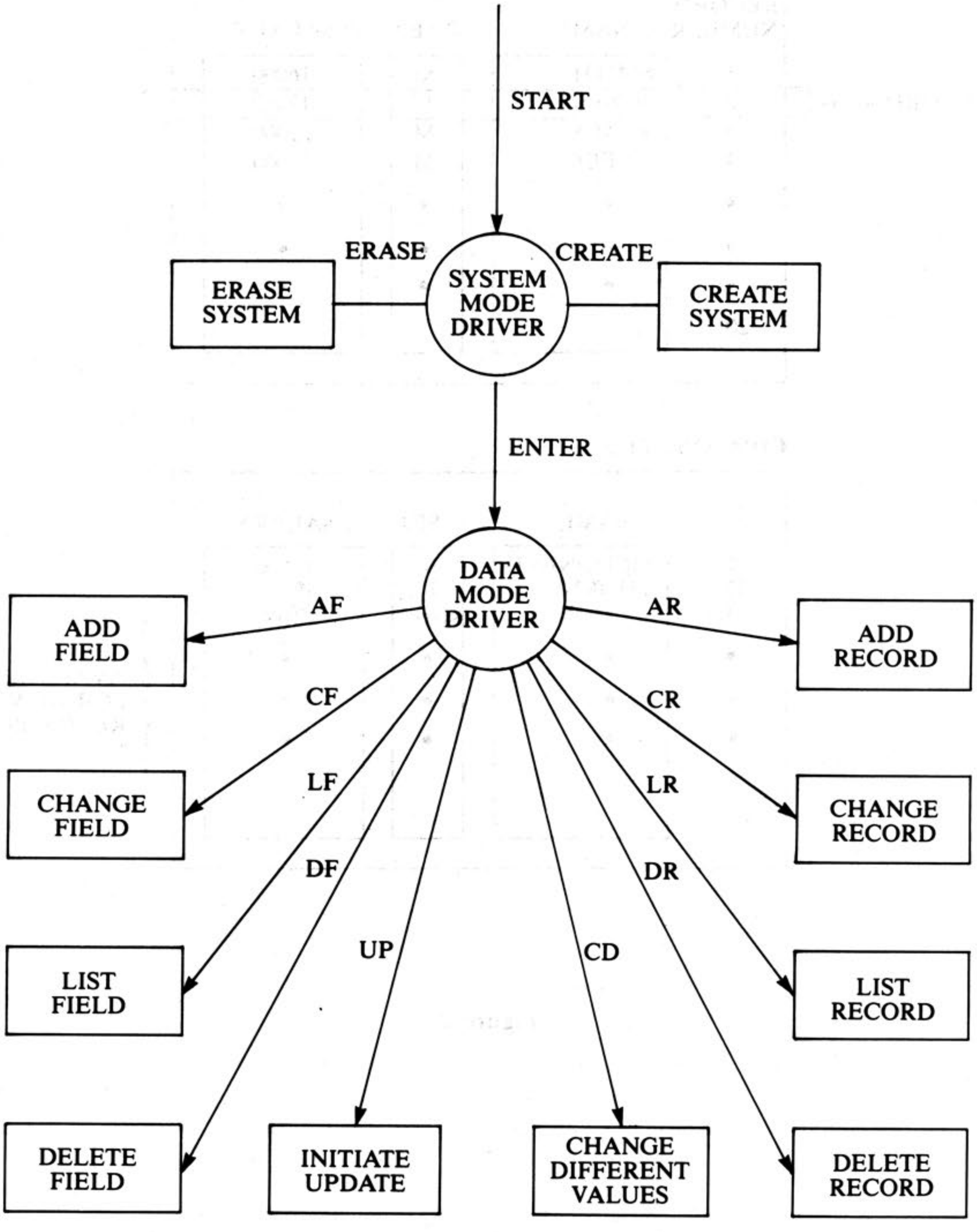

START
ERASE
CREATE
ERASE SYSTEM
SYSTEM MODE DRIVER
CREATE SYSTEM
ENTER
DATA MODE DRIVER
AF
AR
ADD FIELD
ADD RECORD
CF
CR
CHANGE FIELD
CHANGE RECORD
LF
LR
DF
DR
UP
CD
LIST FIELD
LIST RECORD
DELETE FIELD
INITIATE UPDATE
CHANGE DIFFERENT VALUES
DELETE RECORD

Figure 3

A DERIVATIVE ALGORITHM FOR APL

D.M. DeTurck
Courant Institute of Mathematical Sciences
New York University
New York, New York
and
D.L. Orth
IBM T.J. Watson Research Center
Yorktown Heights, New York

Introduction

The purpose of this paper is to illustrate an effective algorithm for computing derivatives in APL. A detailed derivation will appear in [1]. Since derivatives are important components of many algorithms, as well as fundamental tools for analyzing functions, a derivative algorithm will provide many APL users with a useful capability.

Derivative algorithms are not uncommon, and conventional approaches to these algorithms are briefly outlined in Sections 1 and 2. However, the conventional algorithms have been used only for classes of functions that are much less general than the class of APL expressions, and appear to us to be unsuitable for this wider class. Our algorithm (Sections 3 and 4), which is based on Taylor polynomials, applies to all APL expressions, and while it is apparently a new approach to a derivative algorithm, its theoretical basis can be found in most introductory calculus texts.

What, in fact, is a derivative? The concept can be described in terms of tangent lines and tangent planes to graphs of functions, but this description is limited to functions whose graphs can be drawn, such as scalar-valued functions that apply to vectors of length at most two. Thus, while graphical representations are helpful, they do not convey the appropriate generality of the concept, particularly to APL users. A more appropriate description is as follows:

> The derivative of the monadic function F is also a monadic function, denoted here by $\underline{D}F$, that obeys the following **rule of shapes**:
>
> If F applies to arrays of shape S and produces arrays of shape T, then $\underline{D}F$ also applies to arrays of shape S, but produces arrays of shape S,T. (For example, if ×\⍵ takes vectors as arguments then its results are vectors, and therefore if its derivative is applied to vectors it will produce matrices.)
>
> The derivative $\underline{D}F$ provides, at each argument A, a linear approximation to the function F for arguments near A. That is, if the values of F are scalars or vectors then the function values $F\ A+H$ are approximately equal to

```
(F A)+H+.×D̲F A
```

in the sense that the error

```
(F A+H)-(F A)+H+.×D̲F A    N
```

is on the order of H+.×H for small values of H. More generally, the function values F A+H are approximately equal to

```
(F A)+(,H)+.×((×/ρA),ρF A)ρD̲F A
```

The value D̲F A can be defined in terms of the so-called **partial derivatives**. We will speak of the I th **element of the array** A to mean

```
A[I[0];I[1];...;I[N-1]]
```

where N is the rank of A, and will call I an **index** of A. We will also say that the array D is the I th **Kronecker array of shape** S if ρD equals S, all elements of D except the I th equal 0, and the I th element equals 1. Then the I th **partial derivative of** F **at** A, if it exists, is that value which is approached by values of the difference-quotient

```
((F A+T×D)-F A)÷T
```

as values of the scalar T are chosen closer and closer to zero, where D is the I th Kronecker array of shape ρA. The value D̲F A of the derivative at A can then be defined as a collection of all partial derivative values at A; namely, if I is any index of A, then

```
(D̲F A)[I[0];I[1];...;I[N-1];;...;]
```

equals the value of the I th partial derivative of F at A.

If F is nonscalar-valued and if J is an index of the values of Fω, then the J th **component function** of F is defined to be

```
(Fω)[J[0];J[1];...;J[M-1]]
```

where M equals ρρFω. The derivative of the J th component function of F is identical to

```
(D̲Fω)[;;...;J[0];J[1];...;J[M-1]]
```

and therefore this expression is said to define the J th **component of the derivative** of F.

Derivatives of dyadic functions can be described in terms of derivatives of monadic functions, but we will restrict our attention to monadic functions.

The derivative of the derivative of *F*, denoted by *D̲D̲F*, is called the **second derivative** of *F*, the derivative of *D̲D̲F* is called the **third derivative** of *F*, and so on. In this regard *D̲F* is called the **first derivative** of *F* and *F* is called the **zeroth derivative** of itself. We speak of the **higher derivatives** of *F* to mean the second, third, etc. derivatives of *F*, and we also call the *N*th derivative of *F* the *N*th-**order derivative** of *F*.

The domain of a function contains the domain of its derivative, but it is possible that an argument of a function is not in the domain of its derivative. For example, 0 is not in the domain of the derivative of the signum function ×⍵. We therefore say that a function is **differentiable at an argument** *A* if *A* is in the domain of its derivative, or we simply say that it is **differentiable** to mean that its derivative is defined wherever the subsequent discussion requires it to be defined.

One aspect of the derivative algorithm that APL users may like to explore is the way scalar extension is handled, and more generally, the interplay with the axis operator. APL distinguishes between scalar and nonscalar functions, while calculus distinguishes between functions of one variable (**univariate** functions) and functions of more than one variable (**multivariate** functions). All nonscalar functions are multivariate functions, as are all scalar functions when applied to nonscalar arguments. Only functions that are applied to scalars and produce scalars are univariate functions. That the univariate-multivariate classification is an appropriate one for derivatives can be seen from the above rule of shapes. However, distinguishing scalar functions applied to scalars from scalar functions applied to nonscalars is quite uncommon in APL applications.

Section 1 concerns symbolic differentiation, which is a process by which an expression for a derivative is derived from the expression for its source. The topic of Section 2 is numerical differentiation, which is a collection of procedures for producing approximate values of derivatives without using expressions for those derivatives. Sections 3 and 4 deal with Taylor polynomials, which provide a way for evaluating, all at once, as many derivatives of different orders as desired. Section 3 is restricted to univariate functions and parallels Section 1. Section 4 outlines the extension of the results of Section 3 to multivariate functions.

Iverson's direct form of function definition ([2]) is used in this paper. The index origin is 0.

1. Symbolic Differentiation

The algorithm presented here formalizes the way derivatives are ordinarily computed by hand. We do not intend to pursue a complete description of the algorithm, but will simply indicate how such an algorithm can be defined, and then indicate why an APL symbolic derivative algorithm of sufficient generality is impractical. A fairly complete description of symbolic differentiation for monadic scalar functions can be found in [3], with some of the more mathematical parts left to exercises. For our purposes here it is enough to consider a single example, say, the monadic scalar function

```
      (⋆3+⍵)×2○⍵                                   1.1
```

For every primitive function in this expression there is an associated **derivative rule** to be applied to some specific part of the expression, the final result being the derivative. The order of application of the rules is not arbitrary, but, before we discuss it, we will present the individual rules for the primitive functions in 1.1.

For every dyadic scalar function f, and for every pair of differentiable monadic scalar functions *F* and *G*, the monadic scalar function

```
(Fω)f(Gω)
```

has a derivative, and we speak of the **dyadic f rule for derivatives** as the expression which describes that derivative in terms of f, *F*ω, *G*ω, the derivatives of *F*ω and *G*ω, and possibly other primitive (scalar) functions. For example, the **dyadic + rule for derivatives** states that the derivative of (*F*ω)+(*G*ω) is

```
(D̲Fω)+(D̲Gω)                                    1.2
```

where D̲*F*ω and D̲*G*ω denote the derivatives of *F*ω and *G*ω respectively, while the **dyadic × rule for derivatives** states that the derivative of (*F*ω)×(*G*ω) is

```
((D̲Fω)×Gω)+(Fω)×D̲Gω                            1.3
```

Usually one does not speak of rules for derivatives of monadic scalar functions, but it is very convenient to do so. For the example at hand (1.1), we will need the **monadic * rule for derivatives** and the **monadic 2○ rule for derivatives**, which state that the derivatives of **F*ω and 2○*F*ω are

```
(*Fω)×D̲Fω        and        (-1○Fω)×D̲Fω          1.4
```

respectively. Note that the chain rule for derivatives has been incorporated in each of the rules.

The mechanics of symbolic differentiation by which the individual rules are applied are straightforward, the only fairly difficult part being the determination of the **principal function** of an APL expression, i.e. that function which is evaluated last whenever the expression is evaluated. For example, the principal function of the expression (*3+ω)×2○ω is times. To differentiate an expression, one first applies the derivative rule associated with its principal function. In the example at hand, the times rule (1.3) is applied first. Therefore, if the function D̲ applies to a character vector representing a monadic scalar function and produces a character vector representing the derivative of its argument, then the following two expressions are equivalent:

```
D̲ '(*3+ω)×2○ω'
'((',(D̲ '*3+ω'),')×2○ω)+(*3+ω)×',D̲ '2○ω'
```

The next step is to determine the derivatives of *3+ω and 2○ω, whose principal functions are *ω and 2○ω respectively. According to the monadic * and 2○ rules (1.4), D̲ '*3+ω' is equivalent to '(*3+ω)×',D̲ '3+ω' and D̲ '2○ω' is equivalent to '(-1○ω)×',D̲ 'ω', and therefore the following are equivalent:

```
D̲ '(*3+ω)×2○ω'
'(((*3+ω)×',(D̲ '3+ω'),')×2○ω)+(*3+ω)×(-1○ω)×',D̲ 'ω'
```

Finally, according to the plus rule (1.2), D̲ '3+ω' is equivalent to (D̲ '3')+D̲ 'ω'; but D̲ '3' is '0', as is D̲ applied to any character vector representing a constant, and D̲ 'ω' is '1', and so the following are equivalent:

```
D̲ '(*3+ω)×2○ω'
'(((*3+ω)×0+1)×2○ω)+(*3+ω)×(-1○ω)×1'          1.5
```

Evidently, the production of a symbolic derivative is a recursive procedure that terminates with the derivatives of constants (value `0`) or the derivative of the expression *ω* (value `1`).

Expressions for derivatives are often more complex than the original expressions. For example, the second expression in 1.5 can be simplified somewhat, to, say:

```
        '(*3+ω)×(2○ω)-1○ω'                                  1.6
```

but even so, it is more complicated than 1.1. The increased complexity of expressions for derivatives is more pronounced for nonscalar functions. For example, the derivative of `×/ω` is `ω×.*(ι¯1↑ρω)∘.≠ι¯1↑ρω`, while the expressions for the derivatives of `*/ω` and `×/2○ω` are even more complex. Furthermore, if a derivative is more complex than its source, then the higher derivatives are, in general, increasingly complex. Thus, symbolic differentiation can be impractical because it can lead to extremely complicated constructions.

Nevertheless, it should be noted that of all the primitive monadic scalar APL functions, only the Gamma function `!ω` has a derivative that cannot be expressed in terms of primitive functions. Therefore only the Gamma function would be outside the domain of a primitive symbolic derivative operator (because no derivative rule can be formulated).

2. Numerical Differentiation

Perhaps the simplest way to evaluate the derivative of a monadic scalar function *F* is to approximate it with values of the difference-quotient `((F ω+H)-F ω)÷H`. However, it is difficult to obtain uniformly accurate results from difference-quotients because it is often difficult to determine an appropriate value of *H*. Moreover, when *H* is small, as it theoretically must be for accurate approximations, computational accuracy is lost in the subtraction, and therefore even more accuracy will be lost when higher derivatives are evaluated. Thus we should try the other standard approach to numerical differentiation, that being approximating polynomials.

There are two formally equivalent ways of defining a function, one as an expression, or series of expressions like an APL function definition, and the other as a table whose first column holds all the function arguments and whose second column holds the corresponding values. Evidently such tables would often be infinite, but even when finite they are usually not a practical way to represent functions. However, it is often possible to choose a manageable number of representative arguments and values which, when collected in a table, do make an effective, but approximate, representation of the function. In fact, in many cases the values in such a function table are determined not by a formal procedure but by experimentation, such as for a function that represents measurements of some physical quantity.

It may happen that a function definition is needed to analyze a function represented by an approximating table, in which case an approximate definition can usually be constructed from fitting polynomials. In particular, numerical integrals and derivatives of the function are taken to be the corresponding integrals and derivatives of an approximating polynomial.

There are, then, two potential sources of errors in numerical integration and differentiation, one being the approximate definition of the function and the other the approximate function values in the table. While numerical integration tends to smooth the error due to approximate function values, numerical differentiation tends to magnify it. Even when function values are exact and the only source of error is due to approximate function definitions, there are problems similar to those with difference-quotients: it is difficult to obtain uniformly accurate results for first derivatives, and the numerical values tend to be less accurate for higher-order derivatives. Thus texts on numerical analysis usually warn readers to apply numerical differentiation with care.

Numerical differentiation, then, is not a serious candidate for a general purpose APL derivative algorithm. However, it is the only way to provide the derivative of the Gamma function `!ω`.

3. Taylor Polynomials for Univariate Functions

There is a third approach to differentiation, one based on **Taylor polynomials**, which we feel is the best choice even though it has rarely been used. The reason for its unpopularity may be its restriction to algebraic and elementary transcendental functions, or the difficulties in describing the algorithm and applying its results, especially for multivariate functions. In the case of APL, however, only the Gamma function `!ω` is not an algebraic or elementary transcendental function (and is therefore excluded from the present considerations), while the description of the algorithm and its application are straightforward. The advantages of the Taylor polynomial approach are that one application of the algorithm can produce values of as many derivatives of different orders as desired, and the extension of the algorithm from univariate functions to multivariate functions is not difficult.

This is not the appropriate place to go into the technical derivations that support our proposed algorithm, but we can state the conclusions in a way that permits illustration of the algorithm and allows comparison with the symbolic algorithm. The conclusion is this:

3.1 Let *E*: `(α+1)↑ω,1`. Then for every differentiable monadic univariate function *F* there is an associated monadic function *TF* such that for a non-negative integer *N* and an argument *A* of *F*, the value *TF N E A* is a vector of length *N*`+1` whose first element is the value *F A*, whose second element is the value *DF A* of the derivative of *F*, whose third element is the value *DDF A* of the second derivative of *F*, divided by `!2`, and so on. The value *TF N E A* is the **coefficient vector of the** ***N*****th—order Taylor polynomial of** *F* **at** *A*. (Note that *N E A* is the coefficient vector of the *N*th order Taylor polynomial of the identity function `ω` at *A*.)

The associated functions can be produced in either of two ways, which we call **alternate expression** and **alternate evaluation**. Alternate expression is similar to symbolic differentiation, in that new expressions are produced. Unlike symbolic differentiation, however, a new expression produced here is no more complex than its source, in that it is obtained by a one-for-one replacement of each function name and symbol in the source with the name of its associated function. Moreover, the new expression can produce values of derivatives of any desired orders, while symbolic differentiation must be repeated as many times as the order of the highest desired derivative. The alternate expression method will be illustrated with the same example as used in Section 1, namely `(*3+ω)×2○ω` (1.1).

In Section 1, the derivative of 1.1 was produced by applying derivative rules associated with the primitive functions. The alternate expression method produces the associated function of 1.1 simply by replacing each primitive function symbol with the name of its associated function. The order of replacement is immaterial. The definitions of the associated functions for plus and times are based on polynomial arithmetic (see [1-3]), and are

```
S̲: (M↑α)+(M←(ρα)⌈ρω)↑ω                      3.2
```

and

```
P̲: ((ρα)⌈ρω)↑+⌿(-⍳ρα)⌽α∘.×ω,(¯1+ρα)ρ0       3.3
```

respectively. The associated functions of ⋆ω and 2○ω, and of all monadic scalar functions in general, are the Taylor coefficient functions of Paragraph 3.1. In order to define these functions we must first define a function C̲ with the property that the associated function of the composite function *F Gω* is (T̲F Gω)C̲(T̲Gω). The definition of C̲ is as follows (a derivation can be found in [3]):

```
C̲: αSUBS 0,1↓ω                               3.4

SUBS: (ATS⍉((¯1↑ρα),ρω)ρω)+.×α
ATS: (ATS 0 ¯1↓ω)APP PTRω : 0=¯1↑ρω : 1 1ρ1
APP: (((ρω),¯1↑ρα)↑α),ω
PTR: α[;0]P̲ PTR 0 1↓α : 0=¯1↑ρα : 1ρ1
```

Returning to the associated functions of ⋆ω and 2○ω, the coefficient vectors of the Taylor polynomials of ⋆ω and 2○ω are

```
(⋆ω)÷!⍳α+1
```

and

```
((α+1)ρ1 ¯1 ¯1 1×4ρ2 1○ω)÷!⍳α+1
```

respectively, and therefore the associated functions of ⋆ω and 2○ω are

```
T̲S: ((⋆1↑ω)÷!⍳ρω)C̲ ω                         3.5
```

and

```
T̲C: (((ρω)ρ1 ¯1 ¯1 1×4ρ2 1○1↑ω)÷!⍳α+1)C̲ ω
```

respectively. Thus if the function T̲ applies to a character vector representing a monadic function and produces a character vector representing the associated function of its argument, as described in Paragraph 3.1, we have the following equivalent expressions:

```
T̲ '(⋆3+ω)×2○ω'
'(T̲S 3 S̲ω)P̲ T̲Cω'
```

Furthermore, if *F*: (⋆3+ω)×2○ω and T̲F: (T̲S 3 S̲ω)P̲ T̲Cω, then, for example, the values of the third, fifth, and twelfth derivatives of *F* at 3.71 are

```
(12 T̲F 3.71)[3 5 12]×!3 5 12
```

Both symbolic differentiation and the alternate expression method illustrated above can be used to define operators which take differentiable monadic functions as arguments and produce derivative functions as results. In contrast, the alternate evaluation method produces values of the associated functions defined in 3.1 but does not produce the functions themselves. For example, to evaluate the first *N* derivatives of the function *F*: (⋆3+⍵)×2○⍵ for the scalar argument *A* we must evaluate *T̲F N E A*, and the latter evaluation can be accomplished by evaluating *F* in the usual manner, but with the following changes:

(i) the argument *A* is replaced by *N E A*;

(ii) as each function in the definition of *F* is encountered, it is not evaluated, but is replaced by its associated function, and that function is evaluated.

Scalar functions apply element-by-element to nonscalar arguments. The analogous behavior for the associated functions is to apply to vectors along, say, the last axis of non-vector arguments. If the associated functions are so extended, then the function *E* must be replaced with *E*: ((⍴⍵),⍺)↑⍵,[¯0.5+⍴⍴⍵] 1. Be aware that scalar functions applied to nonscalar arguments are multivariate functions, and the extended associated functions are not directly related to the derivatives of those multivariate functions. Consequently the associated functions should be so extended only if the algorithm is to be restricted to scalar functions.

4. Taylor Polynomials for Multivariate Functions

The results of Section 3 extend in a fairly direct manner to the wider class of scalar-valued multivariate functions, such as ×/⋆⍵. According to the rule of shapes stated in the introduction, if *F* is scalar-valued then

⍴*D̲F*⍵	equals	⍴⍵
⍴*D̲D̲F*⍵	equals	(⍴⍵),⍴⍵

and so on, where *D̲F* denotes the derivative of *F* and *D̲D̲F* denotes the second derivative of *F*. In particular, the derivatives of *F* are array-valued, even though *F* itself is scalar-valued. Thus in order to extend 3.1 conveniently, it is useful to permit the elements of the vectors produced by the associated functions to be nonscalar arrays as well as scalars. There are proposed extensions to APL — usually called general array extensions — that permit nonscalar elements in arrays, and for convenience we will use some of them here. In particular, we will assume that:

4.1 (i) the elements of arrays can be nonscalar arrays as well as scalars;

(ii) the scalar functions apply element-by-element to their arguments as they do now, and when an element of an argument is nonscalar, they apply element-by-element to that element as well. For example, if the scalar 10 is added to the two-element vector whose first element is the scalar 1 and whose second element is the vector 2 3, the result is a two-element vector whose first element is the scalar 11 and whose second element is the vector 12 13;

(iii) the operators apply to all functions — primitive, derived, and defined. For example, if *V* and *W* are vectors whose elements are matrices, then *V*∘.(+.×)*W* is a matrix whose *I*,*J*th element is the matrix produced by the +.× inner product of the *I*th element of *V* and the *J*th element of *W*;

(iv) the selection functions apply to right argument arrays whose elements may be nonscalars. For example, if *V* is a three-element vector then `1 0 1/V` is a two-element vector consisting of the first and third elements of *V*, whether those elements are scalars or nonscalars.

Note that these assumptions suggest a new approach to extending the associated functions of Section 3 so that they apply to families of vectors instead of individual vectors. In the last paragraph of Section 3 it was suggested that the functions be applied to vectors along the last axis of their arguments. Another possibility is to apply them element-by-element to all arguments, where the elements would be like the individual vector arguments to which they are now applied. The function *E* must be extended accordingly. As in Section 3, these extensions of the associated functions should be applied only if the algorithm is to be restricted to scalar functions.

Using the proposed language extensions of Paragraph 4.1, we can restate Paragraph 3.1 for scalar-valued multivariate functions. For that purpose it may be helpful to review the paragraph of the introduction that deals with partial derivatives. Then in place of Paragraph 3.1 we have:

4.2 Let *E* be the dyadic function whose value `V←N E A` has the same shape as *A*, and where the *I*th element of *V* is a vector of length `N+1` whose first element is the *I*th element of *A*, whose second element is the *I*th Kronecker array of shape `ρA`, and whose other elements are zero. Then for every differentiable monadic scalar-valued multivariate function *F* there is an associated monadic function *TF* whose value `TF N E A` is a vector containing values of the derivatives of *F* at *A*, as described in Paragraph 3.1. (In particular, the value `TF N E A` is a vector whose first element is a scalar, whose second has shape `ρA`, whose third element has shape `(ρA),ρA`, and so on.)

The alternate expression method carries over to multivariate functions, but not without extending the functions associated with the primitive scalar functions (the required extensions are not those discussed following Paragraph 4.1). For example, consider the scalar-valued multivariate function `×/*ω`. In view of what has been said so far, one should expect that its associated function is `P/TS αEω`, where *E* is defined as in Paragraph 4.2. Evidently *TS* must be extended to accept values of *E* as its argument. By examining several multivariate expressions, we are led to the following general requirement:

> The associated functions of the primitive scalar functions must be extended to apply element-by-element to their arguments, where the elements are vectors of the form `TF N E A`, as described in Paragraph 4.2.

Since it is not difficult to extend functions to apply element-by-element once they are defined for the individual elements, we will concentrate on the part of the extension concerning the appropriate vectors.

For example, consider *TS*, as defined in 3.5. Assuming the language extensions described in Paragraph 4.1, *TS* will automatically extend to the appropriate vector arguments when the definition of *C* is extended. Moreover, the definition of *C* will automatically extend when the definition of *P* is extended. Finally, the extended version of *P* is as follows, where the only difference from 3.1 is the outer product:

```
P: ((ρα)⌈ρω)↑+⌿(-⍳ρα)⌽α∘.(∘.×)ω,(¯1+ρα)ρ0
```

(The definition of $\underline{S}$ in 3.2 does not have to be changed.)

Once the appropriate extensions are made to accommodate scalar-valued multivariate functions, no further extensions are required for nonscalar-valued multivariate functions. For example, $\times/\star\omega$ is scalar-valued if its arguments are scalars or vectors and nonscalar-valued otherwise, and if the above extensions are made so that $\underline{P}/\underline{T}S\ \alpha E\omega$ is defined in the scalar-valued case, then it will also be defined in the nonscalar-valued case. Consequently every monadic function can now have an associated function. The one remaining point is to obtain the derivative of a nonscalar-valued monadic multivariate function from its associated function, and that is done as follows:

> If F is a nonscalar-valued monadic multivariate function with the associated function $\underline{T}F$, then $\rho\underline{T}F\alpha E\omega$ equals $\rho F\omega$, and the Jth element of a result $\underline{T}F\ N\ E\ A$ is a vector whose first element is the Jth element of $F\ A$, and whose second element equals the value of the Jth component of the derivative of F at A (see the description of the derivative in the introduction). The value of the derivative of F at A is therefore obtained by arranging, along the first $\rho\rho A$ axes of an array of shape $(\rho A),\rho F\ A$, the second elements of all the elements of $\underline{T}F\ N\ E\ A$.

References

1. DeTurck, D.M., and D.L. Orth, A Derivative Algorithm Based on Taylor Polynomials, in preparation.

2. Iverson, K.E., **Elementary Analysis**, APL Press, Palo Alto, CA, 1976.

3. Orth, D.L., **Calculus in a new key**, APL Press, Palo Alto, CA, 1976.

PREPARING TO FORECAST

Dinos Appla
I.P. Sharp Associates Limited
Gloucester, England

Abstract

This paper details some of the steps that can be taken in preparing to forecast a time series. It is assumed that the forecaster knows how the forecasts are to be used but very little about the time series itself. A picture is built up of the series — whether it has a trend, is seasonal, is noisy. The magnitudes of the constituent components can be quantified. Achievable goals can be defined. Finally appropriate forecasting methods can be deduced.

Introduction

1) Setting the Scene

In a variety of practical situations the forecaster is faced with the task of forecasting many time series. Effort should be directed at finding the most "appropriate" forecasting method(s) for the series. Grouping and selection can reduce the number of series for individual attention. The choice of a forecasting method depends on the answers to some, or all of the following questions, posed by Chatfield and Prothero [1].

(a) How is the forecast to be used?
(b) What is the degree of required accuracy?
(c) Can external factors be ignored within the context of (a)?
(d) What is known about the structure of the data?
(e) How noisy is the series?
(f) How much history is relevant to the future?

The answers to (a), (b) and (c) should be known a priori.

(a) leads to the 'cost' of errors and a suitable measure of accuracy.
(b) establishes an acceptable level of performance.
(c) decides between a 'causal' or 'extrapolative' approach.

This paper is restricted to 'quantitative extrapolative' methods only.

2) Time Series Models

The answers to questions (d), (e) and (f) may not be known, and some preliminary analysis may be required. To do this, a model needs to be hypothesised for the behaviour of the time series. There are several commonly used models:

Kendall [2] lists the following simple models:

(1) Additive: $x(t)=m(t)+s(t)+e(t)$
(2) Multiplicative: $x(t)=m(t)s(t)e(t)$
(3) Mixed: $x(t)=m(t)s(t)+e(t)$.

Where t indicates time x indicates series
m indciates trend-cycle
s indicates seasonal term
e indicates error term.

More sophisticated than the preceding models the Autoregressive Integrated Moving Average <ARIMA> model of Box and Jenkins [3] and the Dynamic Linear Model <DLM> used by Harrison and Stevens [4].

Preliminary Analysis

1) Graphical

Plotting the observations is not a prerequisite for forecasting, but a graph can give a quick visual impression of any patterns in the data. The graph itself may be quite rough, since it only provides qualitative information.

Series G, shown in EXHIBIT 1.1, is a section of a series of monthly totals of international airline passengers for the years 1949 to 1960 (see [3]). Clearly, there is a strong trend and seasonal component. The seasonal fluctuations increase with level, suggesting the use of methods which assume a multiplicative model. An alternate approach is to take the log of the series and use methods which assume an additive model (see later section on transformations). The regularity of the profile from year to year suggests that the noise levels are not too high.

Series E, shown in EXHIBIT 1.2, is the monthly net income after taxes of Eagle Airlines, from January 1970 to November 1974, from Makridakis and Wheelwright [5]. There is no obvious trend in the data. A seasonal pattern, which is to be expected of airline data, is barely discernible. There seems to be a very high level of noise; external factors may be too important to ignore.

Series X, shown in EXHIBIT 1.3, are the monthly sales of a pharmaceutical product. There is no noticeable trend. There is a seasonal profile with sometimes one or two peaks; the peak of 1956 appears to be exceptionally high.

Series I, shown in EXHIBIT 1.4., displays the total personal disposable income in the U.S. from 1969 to 1978. The series has a trend.

2) Replacement of Extreme Values

Most forecasting methods are not designed to cope with 'dirty' data. Extreme values can arise from errors in recording and can be caused by unusual or unexpected events. Whatever the cause, they should, if possible, be replaced by more representative values. Extreme values may be spotted by scanning a table of the observations and their averages. Such a table is shown in EXHIBIT 2.1 for Series X. Automatic procedures exist for identifying extreme values and calculating their replacements. EXHIBIT 2.2 shows the results of a method suggested by Harrison [6].

3) Error

Both the randomness in a series and the appropriateness of the method contribute to the observed forecast error. For example, consider a series that is generated by the additive process (1), where e(t) is a purely random process. A forecasting method will, at best, produce good estimates of future values of the trend-cycle m(t) and the seasonal component s(t). Let f = M + S, be such a forecast for x at some time t. The observed error, or residual r, is given by,

$$r = x - F = ((m + s) - (M + S)) + e$$
$$\text{residuals} = \text{error[method]} + \text{error[random]} \quad (4).$$

The best any forecasting method can do about the residuals is to reduce them to the level of the random component. Some forecasting methods can also produce confidence intervals for the forecasts by estimating the variance of e(t).

Simple decomposition methods can estimate the random component in a series over history; this estimate can serve as an approximation for error[random] in the future. This determines the 'optimal error' or the average size of residuals that can be achieved by the best method.

The optimal error serves as a target; a good forecasting method should come close to the optimal. The error achieved by the simplest of all forecasting methods should serve as a warning; a good forecasting method should do much better. A suitable candidate for the simplest forecasting method is NAIVEI it simply uses the very latest observations as a forecast for the future.

The measure of accuracy or size of error should be related to how the forecast is to be used. Armstrong [7] gives a good summary of 10 different measures. The MSE (mean square error) and MAPE (mean absolute percent error) are recognised as being generally useful.

EXHIBITS 3.1 to 3.4 show the OPTIMAL and NAIVE errors for the series G, log G, E and X. NAIVE2 is similar to NAIVE1 except the latest observation is adjusted by the ratio of the relevant seasonal indices.

4) Series G

Consider EXHIBIT 3.1. This analysis of series G is based on 'classical decomposition' assuming the simple Multiplicative model (2).

The medial average is another way of handling extreme values. The observations for each month are averaged after removing the largest and smallest values.

The first part of the exhibit shows the period to period percent change of each component, providing a rough guide to the relative importance of each.

The percent change of the random component has another valuable property. It is a reasonable approximation of the best MAPE that can be achieved. The 'OPTIMAL' performance figures are hardly ever achieved.

The NAIVE1 and NAIVE2 figures are both derived from 1-step-ahead forecasts. The MSE and MAPE for NAIVE2, are much better than those for NAIVE1. It is certainly worth it to use forecasting methods which take seasonality into account.

The indications are that there is very little randomness in this series. From these figures, one would expect a forecasting method that was 'appropriate' to give MAPE values in the range 4% to 8%.

EXHIBIT 3.2 was obtained by first transforming the observations by a natural logarithm. Naturally if the series follows a multiplicative model, then its logarithms should follow an additive one.

The percent variations are smaller than in EXHIBIT 3.1. The transformation reduces the variation, but the relative importance of the components stays roughly the same.

5) Transformations

Forecasting methods, such as Box and Jenkins, require a series to be 'stationary'.

It is sometimes possible to make the series stationary by differencing (see [3]), or by subtracting a trend curve. There are occasions when differencing is not enough and some non-linear transformation is required. This is the case for series G, which requires a logarithmic transformation.

It is not easy to find the 'right transformation'. Jenkins [8] suggests the use of a range-mean plot. This is obtained by dividing the time series into sub-series and plotting the range against the mean for each sub-series. The range-mean plot is really a scattergram. If the series does not require a transformation, then the range will be independent of the mean. This will appear as a random scatter about a horizontal or vertical line. If the range-mean plot is a scatter about any other straight line, then the required transformation is a logarithm. The range and mean of a transformed series should be independent if the transformation was the 'right' one.

EXHIBIT 4.1 shows the range-mean plots for the series G, log G, E, X and I. It is immediately clear that series G should be transformed, and that series E, X and I need not be.

There are dangers involved in the process; transformations induce biasing. The effect is that what might appear as the best forecast in transformed units, is not necessarily the best forecast in original units. EXHIBIT 4.2 shows the results of forecasting a quarterly summary of series X. A small report is given stating the performance in terms of MPE, MSE and MAPE over history. The method HARM appears 'worst'.

The second part of the exhibit shows the same report after the series and the predicted series are transformed by a natural logarithm. Here, the picture has changed dramatically. The method HARM which was the worst in the original units, is now the 'best'.

6) Series E

An additive model is assumed in the preliminary analysis of series E above, because the series contains negative values and is centered around 0.

There is a great deal of variation in this series. The achievable MAPE figure of around 160% is discouragingly large. This is a case where external factors should, if possible, be taken into account.

If it is not obvious that seasonality is present, then an autocorrelation analysis of the series should be done (see later section on the correlogram).

7) Series X

The relative proportions of the percent variation of the components of series X, shown in EXHIBIT 3.4, are similar to those of series E. The achievable MAPE of around 29% may be small enough to make an extrapolative method worthwhile.

8) Correlogram

The correlogram is a plot of the autocorrelation coefficients of a series against lag. Autocorrelation describes the association or mutual dependence between values of the same variable, but at different time periods.

Autocorrelation coefficients provide important information about the structure and pattern of the data; they can reveal the presence of both seasonality and trend. See [3] or [5].

If the series is purely random, then the observations are independent of one another. Thus, the correlogram of a purely random series is theoretically 0 for each lag. The coefficients used in the correlogram are, of course, approximations of the theoretical values. The correlogram is plotted with confidence bands around 0; values falling within these bands are not statistically different from 0.

EXHIBITS 5.1 and 5.2 show the correlograms for series I and its first difference. The trend in the series is reflected by a trend in the first correlogram. The trend is removed by the first difference. If seasonality were present, then there would have been a large coefficient at lag 4 (quarterly data). Furthermore, the second correlogram only contains one coefficient significantly different from 0. This coefficient is so close to the confidence band that it is fairly safe to assume it to be spurious. Box and Jenkins [3] show how the correlogram of a purely random series is quite likely to contain one or two such values.

The conclusion is, that after a first difference, series I is purely random. In terms of structure, this is equivalent to saying that the series I is generated by the process:

$$z(t)=z(t\text{-}1)+a(t), \text{ where } a(t) \text{ is purely random}$$

Candidates for forecasting methods would be a straight line fit and extrapolation or a simple exponential smoothing method which took account of trend.

Choosing A Forecasting Method

1) Decision Trees

Reid [9] applied 5 different forecasting methods to each of 113 different series to determine the best (MSE) method for each. He then categorised each series according to:

1. Long (>50 obs) or Short (< 50 obs)
2. Seasonal or Non-seasonal
3. Does random component have High or Low variance in the series compared with other variability?
4. Are there Large peaks, non-stationarities or discontinuities or Not?
5. Predicting Long or Short lead times?

Using these 5 binary decisions, he formed a 'decision tree' for choosing a forecasting method.

Two branches of decision tree are shown below:

	1	2	3	4	5	† Method
Series G.	Long	Seasonal	Low	Not	Long	† Box-Jenkins
Series X.	Short	Seasonal	Low	Yes	Short	† Holt-Winters

Kendall [2] suggests that "every practitioner needs his own tree".

2) Decision Tables

Makridakis and Wheelwright [5] make a priori decisions about appropriate forecasting methods by categorising the methods. Each method is checked to see if it possesses (is known to possess or is designed to possess) the following attributes;

(a) Minimum points required
(b) Good for Stationary series
(c) Good for series with Trend
(d) Good for series which are Seasonal
(e) Good for Short predictive lead times
(f) Good for Medium predictive lead times
(g) Good for Long predictive lead times

The idea is that, for each series, the preliminary analysis has established whether the series is stationary, has trend, is seasonal. The number of observations to be used and the time horizon for the forecasts complete the picture. EXHIBIT 6, shows the recommendations made upon the basis of searching a table of methods against attributes.

Acknowledgements

All the exhibits in this paper were produced within an interactive forecasting workspace available on the SHARP APL system.

The inspiration for this work came from the book by Makridakis and Wheelwright [5]. This made it possible to bring together many of the time series forecasting and analysis functions already available on the Sharp system, and to present them in a uniform manner.

References

[1] Chatfield, C. and Prothero, D.L. "Box-Jenkins Seasonal Forecasting: Problems in a Case Study", J.R. Statist Soc A (1973), 136, Part 3.

[2] Kendall, M. "Time-Series". Charles Griffin & Co. Ltd., 1976 (2nd. ed).

[3] Box, G.E.P. and Jenkins, G.M. "Time Series Analysis Forecasting and Control". Holden-Day, 1976 (Revised ed).

[4] Harrison, P.J. and Stevens, C.F. "A Bayesian Approach to Short Term Forecasting", Oprn Res Q. Vol 22, No 4.

[5] Makridakis, S. and Wheelwright, S.C. "Interactive Forecasting", Holden-Day, 1978.

[6] Harrison, P.J. "Short-Term Sales Forecasting", Applied Stats, Vol. 14, No 23 1965.

[7] Armstrong, J.S. "Long-Range Forecasting", John Wiley & Sons, 1978.

[8] Jenkins, G.M. "Practical Experiences with Modelling and Forecasting Time Series", GJP Publications, 1979.

[9] Reid, D.J. "A Comparison of Forecasting Techniques on Economic Time Series". Proc Joint Conf "Forecasting in Action" 21st, April, 1971, OR Soc and Soc Long Range Planning.

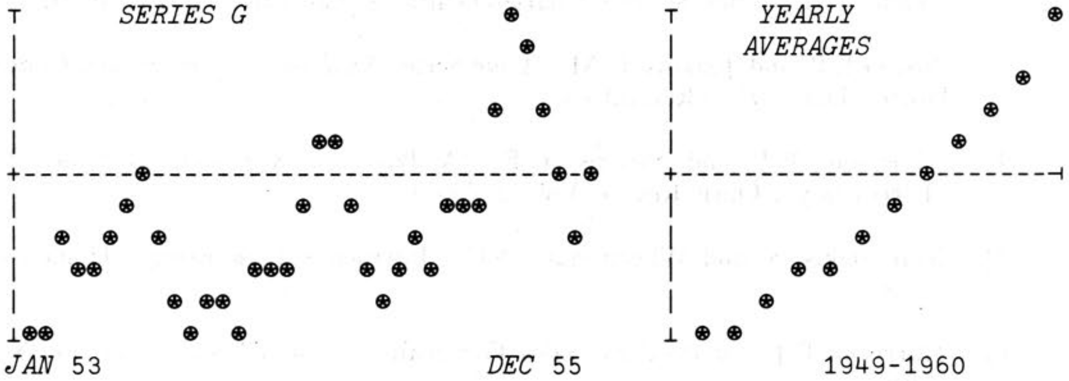


Exhibit 1.1

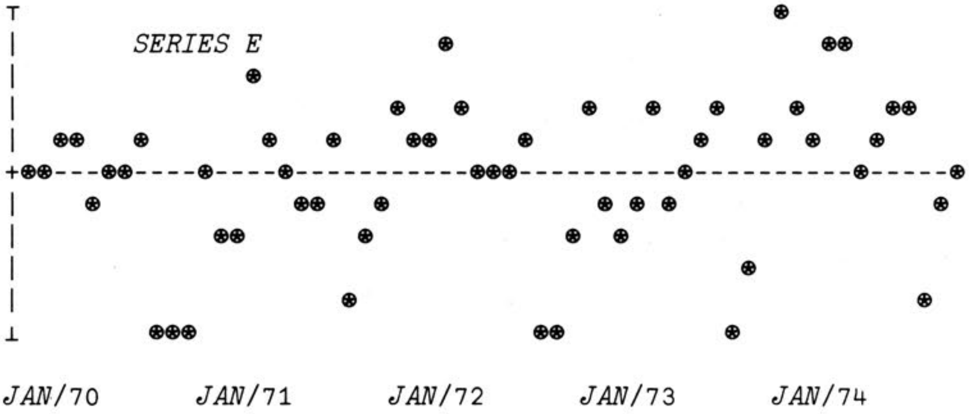


Exhibit 1.2

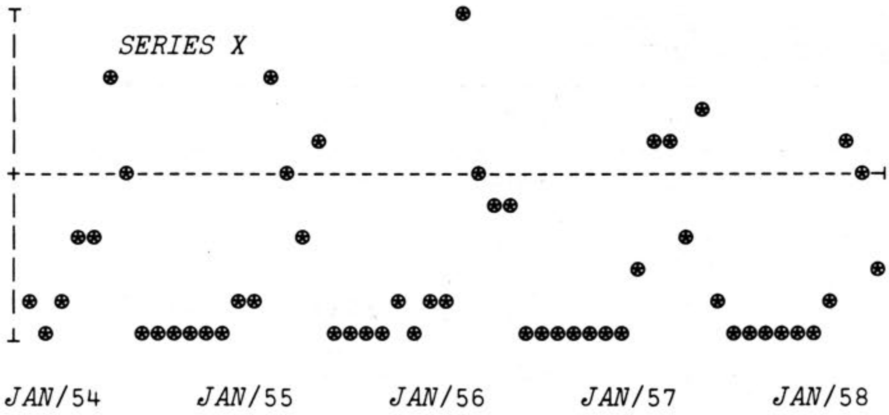


Exhibit 1.3

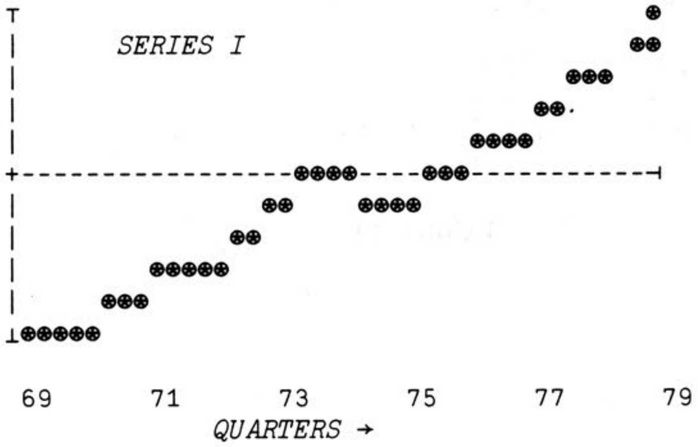


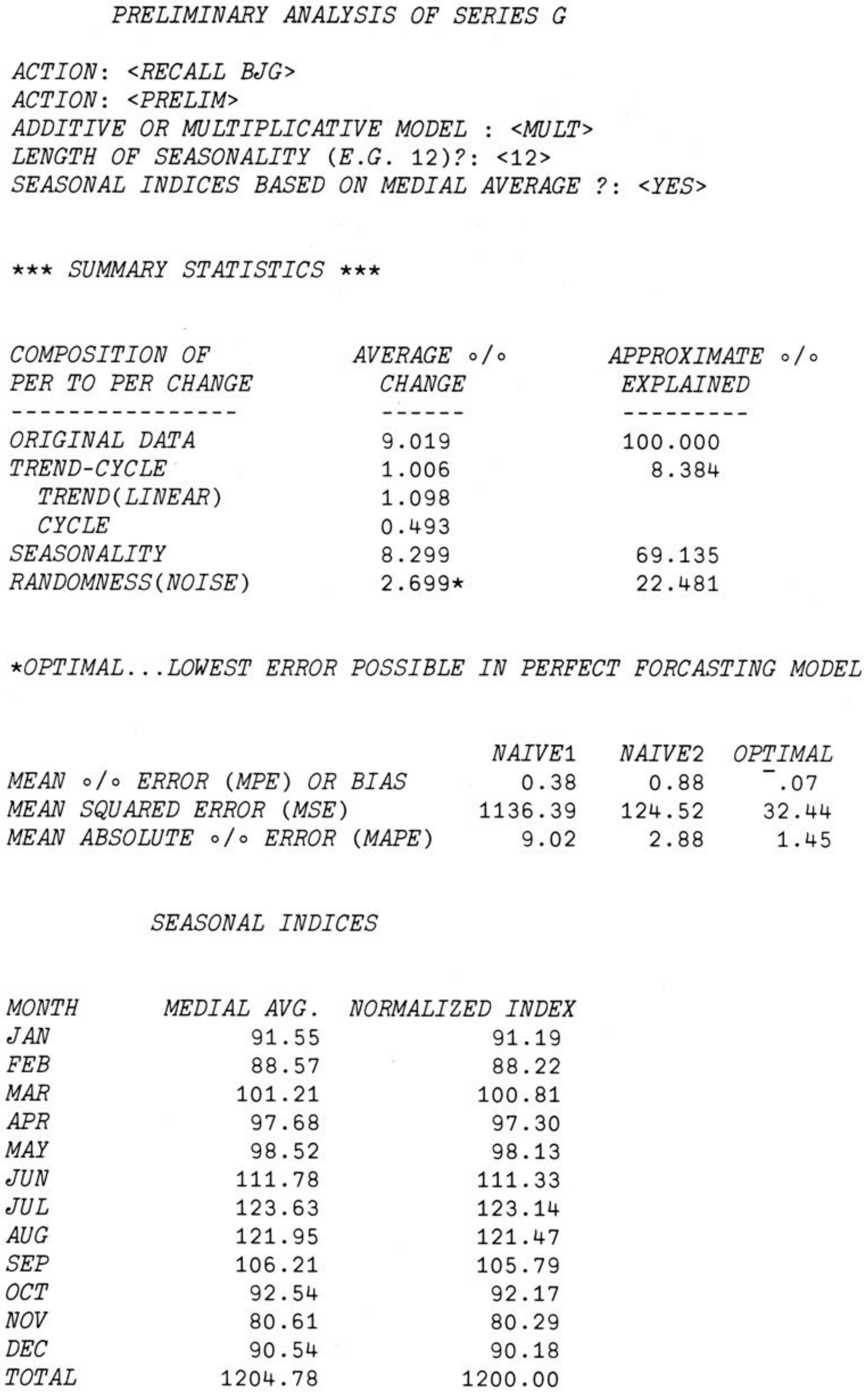
Exhibit 1.4

```
SERIES X, AVERAGED BY MONTH AND YEAR.


ACTION: <RECALL SERIESX>
ACTION: <AVERAGE ACTUALS>
+-------+----------+----------+----------+----------+----------+----------+
|MONTH  |      1954|      1955|      1956|      1957|      1958|       AVG|
+-------+----------+----------+----------+----------+----------+----------+
|JAN    |      6530       3744       3675       4956       3604|      4502|
|       |                                                      |          |
|FEB    |      4917       5614       5589       4909       3401|      4886|
|       |                                                      |          |
|MAR    |      7302       5408       8589       9959       7074|      7666|
|       |                                                      |          |
|APR    |     14720      38079      45430      28425      29631|     31257|
|       |                                                      |          |
|MAY    |     18189      24474      26163      30498      24760|     24817|
|       |                                                      |          |
|JUN    |     40473      14571      21988      17331      11805|     21234|
|       |                                                      |          |
|JUL    |     23604      27932      21214      34835           |     26896|
|       |                                                      |          |
|AUG    |      2391       3314       1532       5875           |      3278|
|       |                                                      |          |
|SEP    |      2884       1772        473       2459           |      1897|
|       |                                                      |          |
|OCT    |      3283       1963       1683       3592           |      2630|
|       |                                                      |          |
|NOV    |      3358       2632       2888       1670           |      2637|
|       |                                                      |          |
|DEC    |      4418       5104       4119       3160           |      4200|
+-------+----------+----------+----------+----------+----------+----------+
|  AVG  |     11006|     11217|     11945|     12306|     13379|     11325|
+-------+----------+----------+----------+----------+----------+----------+
```

Exhibit 2.1

```
         SERIES X,EXTREME VALUES AND REPLACEMENTS.

ACTION: <EXTREME>
ADDITIVE OR MULTIPLICATIVE MODEL : <MULT>
LENGTH OF SEASONALITY: <12>
         REPLACED OUTLIERS
PERIOD      OUTLIER     REPLACEMENT
APR/54     14720.00     28841.18
JUN/54     40473.00     22471.65
APR/56     45430.00     31523.54

              SIGNIFICANT HARMONICS
HARMONIC      COSINE         SINE           AMPLITUDE
    1          0.612         ¯0.913           1.099
    2         ¯0.093         ¯0.414           0.424
    3          0.087          0.203           0.221
    4          0.481          0.052           0.484
    5          0.301         ¯0.186           0.354
```

Exhibit 2.2



```
        PRELIMINARY ANALYSIS OF SERIES G

ACTION: <RECALL BJG>
ACTION: <PRELIM>
ADDITIVE OR MULTIPLICATIVE MODEL : <MULT>
LENGTH OF SEASONALITY (E.G. 12)?: <12>
SEASONAL INDICES BASED ON MEDIAL AVERAGE ?: <YES>


*** SUMMARY STATISTICS ***


COMPOSITION OF          AVERAGE ∘/∘        APPROXIMATE ∘/∘
PER TO PER CHANGE        CHANGE             EXPLAINED
----------------         ------             ---------
ORIGINAL DATA            9.019              100.000
TREND-CYCLE              1.006                8.384
  TREND(LINEAR)          1.098
  CYCLE                  0.493
SEASONALITY              8.299               69.135
RANDOMNESS(NOISE)        2.699*              22.481


*OPTIMAL...LOWEST ERROR POSSIBLE IN PERFECT FORCASTING MODEL


                                     NAIVE1   NAIVE2  OPTIMAL
MEAN ∘/∘ ERROR (MPE) OR BIAS           0.38     0.88     ¯.07
MEAN SQUARED ERROR (MSE)            1136.39   124.52    32.44
MEAN ABSOLUTE ∘/∘ ERROR (MAPE)         9.02     2.88     1.45


          SEASONAL INDICES


MONTH        MEDIAL AVG.  NORMALIZED INDEX
JAN               91.55              91.19
FEB               88.57              88.22
MAR              101.21             100.81
APR               97.68              97.30
MAY               98.52              98.13
JUN              111.78             111.33
JUL              123.63             123.14
AUG              121.95             121.47
SEP              106.21             105.79
OCT               92.54              92.17
NOV               80.61              80.29
DEC               90.54              90.18
TOTAL           1204.78            1200.00
```

Exhibit 3.1

```
PRELIMINARY ANALYSIS OF THE NATURAL LOGARITHM OF SERIES G
ACTION: <RECALL BJG>
ACTION: <TRANSFORM LN>
TRANSFORMING VIA NATURAL LOGARITHM

ACTION: <PRELIM>
ADDITIVE OR MULTIPLICATIVE MODEL : <ADD>

LENGTH OF SEASONALITY (E.G. 12)?: <12>
SEASONAL INDICES BASED ON MEDIAL AVERAGE ?: <Y>


*** SUMMARY STATISTICS ***


COMPOSITION OF           AVERAGE ∘/∘       APPROXIMATE ∘/∘
PER TO PER CHANGE         CHANGE            EXPLAINED
----------------          ------            ---------
ORIGINAL DATA             1.639             100.000
TREND-CYCLE               0.184               8.650
  TREND(LINEAR)           0.182
  CYCLE                   0.073
SEASONALITY               1.451              68.125
RANDOMNESS(NOISE)         0.495*             23.225


*OPTIMAL...LOWEST ERROR POSSIBLE IN PERFECT FORCASTING MODEL


                                      NAIVE1   NAIVE2  OPTIMAL
MEAN ∘/∘ ERROR (MPE) OR BIAS           0.16     0.17
MEAN SQUARED ERROR (MSE)            0.01136  0.00149  0.00037
MEAN ABSOLUTE ∘/∘ ERROR (MAPE)         1.64     0.53     0.27


          SEASONAL INDICES


MONTH      MEDIAL AVG.   NORMALIZED INDEX
JAN        ¯0.07745322     ¯0.08269527
FEB        ¯0.10205382     ¯0.10729587
MAR         0.02410674      0.01886469
APR        ¯0.00984244     ¯0.01508450
MAY        ¯0.00259563     ¯0.00783768
JUN         0.11594462      0.11070257
JUL         0.21493025      0.20968820
AUG         0.20008186      0.19483981
SEP         0.06538807      0.06014602
OCT        ¯0.06907003     ¯0.07431209
NOV        ¯0.20635111     ¯0.21159316
DEC        ¯0.09018067     ¯0.09542273
TOTAL       0.06290462      0.00000000
```

Exhibit 3.2

```
        PRELIMINARY ANALYSIS OF SERIES E

ACTION: <RECALL EAGLE>
ACTION: <PREL>
ADDITIVE OR MULTIPLICATIVE MODEL : <A>
LENGTH OF SEASONALITY (E.G. 12)?: <12>
SEASONAL INDICES BASED ON MEDIAL AVERAGE ?: <NO>


*** SUMMARY STATISTICS ***

COMPOSITION OF           AVERAGE ∘/∘          APPROXIMATE ∘/∘
PER TO PER CHANGE          CHANGE               EXPLAINED
----------------           ------               ---------
ORIGINAL DATA            350.452               100.000
TREND-CYCLE               21.406                 4.190
  TREND(LINEAR)            3.888
  CYCLE                   19.238
SEASONALITY              324.929                63.601
RANDOMNESS(NOISE)        164.550*               32.209


*OPTIMAL...LOWEST ERROR POSSIBLE IN PERFECT FORCASTING MODEL


                                     NAIVE1   NAIVE2  OPTIMAL
MEAN ∘/∘ ERROR (MPE) OR BIAS          22.69    54.71    41.03
MEAN SQUARED ERROR (MSE)               2E7      4E6      1E6
MEAN ABSOLUTE ∘/∘ ERROR (MAPE)       350.45   161.16    68.84


           SEASONAL INDICES


MONTH              AVG.  NORMALIZED INDEX
JAN              999.08           1007.49
FEB             ¯453.06           ¯444.66
MAR             4353.79           4362.19
APR             3633.90           3642.30
MAY             ¯356.19           ¯347.78
JUN             ¯252.67           ¯244.26
JUL              949.56            957.97
AUG             2499.33           2507.74
SEP            ¯7220.71          ¯7212.31
OCT            ¯5790.81          ¯5782.41
NOV            ¯2384.23          ¯2375.83
DEC             3921.17           3929.57
TOTAL           ¯100.83              0.00
```

Exhibit 3.3

```
     PRELIMINARY ANALYSIS OF SERIES X
     AFTER REPLACEMENTS (SEE EXHIBIT 2.2)
ACTION: <PRE>
ADDITIVE OR MULTIPLICATIVE MODEL : <M>
LENGTH OF SEASONALITY (E.G. 12)?: <12>
SEASONAL INDICES BASED ON MEDIAL AVERAGE ?: <NO>

*** SUMMARY STATISTICS ***

COMPOSITION OF            AVERAGE ∘/∘        APPROXIMATE ∘/∘
PER TO PER CHANGE           CHANGE             EXPLAINED
----------------            ------             ---------
ORIGINAL DATA             106.320              100.000
TREND-CYCLE                 2.206                1.786
 TREND(LINEAR)              0.098
 CYCLE                      2.222
SEASONALITY                91.625               74.160
RANDOMNESS(NOISE)          29.719*              24.054


*OPTIMAL...LOWEST ERROR POSSIBLE IN PERFECT FORCASTING MODEL


                                     NAIVE1   NAIVE2  OPTIMAL
MEAN ∘/∘ ERROR (MPE) OR BIAS         ¯63.63    ¯7.67    ¯4.80
MEAN SQUARED ERROR (MSE)               1E8      2E7     4E6
MEAN ABSOLUTE ∘/∘ ERROR (MAPE)       106.32    28.60    15.22
```

Exhibit 3.4

```
                   RANGE-MEAN PLOTS

 T                K L        T                  J K
 |                           |            F         L
 |              J            |              G
 |             I             |                  I
 |                           |           E    H
R|         G H              R|     B
A|       F                  A|
N|                          N|
G|      E                   G|   A
E|     D                    E|         D
 |  B C                      |       C
 |                           |
 +-------------------------| +-------------------------|
          MEAN                        MEAN

           G                          LOG G


 T      D        T   C          T        D  G    I
 |               |              |
 |               |              |
 |               |              |                   J
 |               |   A          |
R|       E      R|   B         R|
A|      B       A|   D         A|
N|              N|             N| A B
G|              G|             G|             H
E|              E|             E|           E
 |     A         |    E         |     C    F
 |               |              |
 +-----------|   +-----------|  +--------------------|
   MEAN            MEAN                MEAN
```

Exhibit 4.1

```
ACTION: ≤RECALL XQ≥
ACTION: ≤LIST CURRENT≥
SERIES <CURRENT> :18 OBSERVATIONS ,16 USED.
TIMEFRAME: QUARTERLY DATED 1 54 TO 2 58
LENGTH OF SEASONALITY 4
ANALYSES PERFORMED WITH STATISTICS AND PERIODS FORECAST (WHERE APPLICABLE)
                         EXPW1     EXPW2      HARM     EXPW3     EXPMW
         MPE            ¯45.82      2.03    ¯72.16     23.34      8.37
         MSE             8E8       7E8       1E9       6E8       6E8
         MAPE           112.51     65.29    117.25     59.10     66.16

PERIODS FORECAST             2         2         2         2         2

*****

ACTION: ≤TRANSFORM LN≥
TRANSFORMING VIA NATURAL LOGARITHM

ACTION: ≤LIST CURRENT≥
SERIES <CURRENT> :18 OBSERVATIONS ,16 USED.
TIMEFRAME: QUARTERLY DATED 1 54 TO 2 58
LENGTH OF SEASONALITY 4
DATA TRANSFORMED VIA:     LN
ANALYSES PERFORMED WITH STATISTICS AND PERIODS FORECAST (WHERE APPLICABLE)
                         EXPW1     EXPW2      HARM     EXPW3     EXPMW
         MPE             27.00     29.12      3.32     30.94     29.77
         MSE             32.13     31.81      6.88     31.80     31.83
         MAPE            35.97     33.95     13.79     33.85     34.05

PERIODS FORECAST             2         2         2         2         2

*****
```

Exhibit 4.2

```
                AUTOCORRELATION ANALYSIS OF SERIES I

              AUTOCORRELATIONS: DIFFERENCING 0

MEAN AUTOCORRELATION          0.436   STANDARD ERROR            0.158
MEAN OF FIRST 7 VALUES        0.643   CHI SQUARE (COMPUTED)   135.065
MEAN OF LAST 6 VALUES         0.194   CHI SQUARE (TABULATED)    5.227
COEFFICIENT OF VARIATION      0.094

DATA DO NOT YET APPEAR STATIONARY
SOME NON-RANDOM PATTERN SEEMS TO EXIST IN THE DATA,
SINCE 8 VALUES FALL OUTSIDE CONTROL LIMITS.

GRAPH THE AUTOCORRELATIONS? : <Y>

    ¯1.0 ¯0.8 ¯0.6 ¯0.4 ¯0.2  0.0  0.2  0.4  0.6  0.8  1.0
LAG  +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+   VALUE
  1                   T       +      T               o     0.904
  2                   T       +      T             o       0.813
  3                   T       +      T          o          0.726
  4                   T       +      T         o           0.637
  5                   T       +      T      o              0.546
  6                   T       +      T    o                0.472
  7                   T       +      T   o                 0.404
  8                   T       +      T o                   0.338
  9                   T       +      T                     0.279
 10                   T       +    o T                     0.224
 11                   T       +   o  T                     0.163
 12                   T       +  o   T                     0.105
 13                   T       + o    T                     0.054
```

Exhibit 5.1

```
                    AUTOCORRELATION ANALYSIS OF SERIES I

                  AUTOCORRELATIONS: DIFFERENCING 1

MEAN AUTOCORRELATION      ¯0.029   STANDARD ERROR            0.160
MEAN OF FIRST 7 VALUES    ¯0.050   CHI SQUARE (COMPUTED)    11.425
MEAN OF LAST 6 VALUES     ¯0.004   CHI SQUARE (TABULATED)    5.227
COEFFICIENT OF VARIATION   0.012

DATA APPEAR STATIONARY
NO SIGNIFICANT NON-RANDOM PATTERN SEEMS TO EXIST.

GRAPH THE AUTOCORRELATIONS? : <Y>

   ¯1.0  ¯0.8  ¯0.6  ¯0.4  ¯0.2   0.0   0.2   0.4   0.6   0.8   1.0
LAG  +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+   VALUE
  1                       T       o+        T                      ¯0.060
  2                       T        +o       T                       0.008
  3                       T        +      o T                       0.225
  4                       T      o +        T                      ¯0.080
  5                     o T        +        T                      ¯0.371
  6                       T        + o      T                       0.056
  7                       T     o  +        T                      ¯0.125
  8                       T     o  +        T                      ¯0.118
  9                       T        + o      T                       0.053
 10                       T        +      o T                       0.204
 11                       T      o +        T                      ¯0.098
 12                       T      o +        T                      ¯0.084
 13                       T        +o       T                       0.016
```

Exhibit 5.2

```
              RECOMMENDED FORECASTING METHODS FOR SERIES I

ACTION: <RECALL INCOME>
ACTION: <RECOMMEND>
USE HOW MANY OF THE 40 OBSERVATIONS?: <ALL>
AUTOCORRELATION ANALYSIS OF YOUR DATA?: <YES>

    [OUTPUT IDENTICAL TO EXHIBITS 5.1 AND 5.2]

                  AUTOCORRELATIONS: DIFFERENCING 2

MEAN AUTOCORRELATION      ¯0.036  STANDARD ERROR           0.162
MEAN OF FIRST 7 VALUES    ¯0.071  CHI SQUARE (COMPUTED)   24.442
MEAN OF LAST 6 VALUES      0.004  CHI SQUARE (TABULATED)   5.227
COEFFICIENT OF VARIATION   0.017

DATA APPEAR STATIONARY
NO SIGNIFICANT NON-RANDOM PATTERN SEEMS TO EXIST.

GRAPH THE AUTOCORRELATIONS? : <NO>


SUMMARY: DATA APPEAR STATIONARY AT DIFFERENCING 1

AUTOCORRELATIONS SIGNIFICANTLY DIFFERENT FROM ZERO:
LAG   AUTOCORRELATION
 5        ¯0.37

FURTHER AUTOCORRELATION ANALYSIS? : <NO>
PRELIMINARY ANALYSIS?: <YES>


ADDITIVE OR MULTIPLICATIVE MODEL : <A>
LENGTH OF SEASONALITY (E.G. 12)?: <0>

*** SUMMARY STATISTICS ***

COMPOSITION OF            AVERAGE ∘/∘         APPROXIMATE ∘/∘
PER TO PER CHANGE           CHANGE              EXPLAINED
-----------------           ------              ---------
ORIGINAL DATA               1.1690               100.000
TREND-CYCLE                 1.001                61.449
 TREND(LINEAR)              0.806
 CYCLE
RANDOMNESS(NOISE)           0.628*               38.551

                                   NAIVE1           OPTIMAL
MEAN ∘/∘ ERROR (MPE) OR BIAS         0.88
MEAN SQUARED ERROR (MSE)           154.17             26.82
MEAN ABSOLUTE ∘/∘ ERROR (MAPE)       1.17              0.45

TIME HORIZON FOR FORECASTS: <3>


SUITABLE FORECASTING METHODS

MODULE             TYPE  COMPLEXITY

CURVEFIT             1      1.5
SMOOTH               3      2.0
SMOOTH               4      2.0
AVERAGE              2      2.0
ARAF                 1      7.5
BOXJENKINS           1     10.0
```

Exhibit 6

MONITORING THE LISTED STOCK OPTION

Howard M. McLean
Merrill Lynch Royal Securities Limited
Toronto, Ontario

Less than eight years ago, as the Chicago Board Options Exchange prepared to launch their "bright idea" new product, many seasoned Wall Street observers predicted (in some cases on paper, to their eventual chagrin) the quick and decisive failure of the Listed Stock Option, whose new concepts and jargon would render acceptance by investors and speculators less than likely.

Listed options were not accepted by the investing and speculating public. They were embraced passionately. In a world where inflation was moving from the arcane vocabulary of economists to the profane vocabulary of its myriad victims, listed options, with their leveraged and accelerated response to changing stock market conditions, seized the imagination of a large segment of the market, and have yet to let go.

Like any new product, listed options moved first through an early "recognition" phase where availability impressed itself on the marketplace; a subsequent period of sharp and expanding growth as the product and its targeted consumers found each other; and finally, more recently, into an apparent "saturation" phase wherein listed option trading volume as a percentage of overall stock market activity appears to have stabilized in a narrow month-to-month band.

In short, the listed option market has reached early middle age, when youth-induced optimism and ebullience are likely to find themselves sharing the psychic stage, on occasion, with bouts of reflection and speculation about the future. And, as is often the case, the retrospective summary is mixed: opportunities seized and opportunities lost; promise fulfilled and promise squandered.

It will be our attempt, in the brief space of this paper: first, to outline what listed options are, and thereby sketch the implied opportunities which have intrigued thoughtful speculators and investors from the beginning; second, to discuss the chimerical promise of "static" options tactics and strategies, and show why this promise was largely an academic mirage; third, to hint briefly at where the true promise of the options market lies, and how it must be developed by the brokerage industry if it is to be fulfilled.

I. What are Listed Options?

Listed stock options come in two types: calls and puts.

A CALL option gives its owner the right to BUY a certain number of shares at a certain price for a certain length of time, e.g., a

JKL Oct 50 Call

enables its buyer to purchase 100 shares of Jeckyl Corp. common stock at a price of $50 per share at any time up until a fixed expiration date in October. On the other side of the trade, the party who sold this option must stand ready to deliver the 100 shares at $50 until the same expiration date, when the obligation to do so ceases.

A PUT option gives its owner the right to SELL a certain number of shares at a certain price for a certain length of time, e.g., an

HYD Jan 40 Put

enables its buyer to sell 100 shares of Hyde Industries common stock at a price of $40 per share at any time up until a fixed expiration date in January. On the other side of the trade, the party who sold this option must stand ready to receive and pay for the 100 shares at $40 until the same expiration date.

How do listed option transactions take place?

Listed option trading takes place on the floor of an options exchange, of which there are currently seven in North America, where facilities are provided for buyers and sellers to announce the price at which they are prepared to do business, and to confirm and consummate ensuing trades — all in a fair and orderly manner. Currently, options contracts are available for trading on almost 300 underlying stocks.

The Strike Price/Expiration Date Matrix

To identify an option as, for instance, an October 45 Put, is to make reference to the October **expiration date**, which gives the option a determinate life span, and the $45 per share **exercise price** which defines its "privilege" price relationship to the price of the underlying stock. These two characteristics define a particular option series and its price as a matrix data point.

Since one secret to successful free marketing has always been to provide your customers with a choice, listed options come, not in a wide range of colours and flavours, but over a range of exercise prices, which are added as required to bracket the underlying stock price as it fluctuates, and new expiration dates, made available to trading as old expiration dates arrive and option series vanish forever. (Remember the Dome Pete Oct 60's of '79... now **that** was a hot option series!)

Pricing the Matrix

Each of these matrix data points must be priced and repriced, during each trading day, as the value of the underlying stock moves about, first this way, then that; now quickly, then slowly... in general behaving as a foraging ant. A few moments spent studying the record of actual option price, or **premium** levels recorded as final trades by the set of option series relating to one underlying stock (Table 1) will reveal, in a "freeze-frame" format, the general principles which underly option pricing:

1. **Intrinsic value, if any.** If an option series provides a current advantage to its owner, namely the ability to buy stock at a lower price than the current market value in the case of a call, or the ability to sell stock at a higher price than that available in the current market in the case of a put, it has **intrinsic** value. e.g.,

The intrinsic value of a call with an exercise price of $50, when the underlying stock is trading at $53, is clearly $3.

The intrinsic value of a put with an exercise price of $40, when the underlying stock is trading $36, is $4.

If an option series has no current advantage, for instance, a $50 call when the stock is trading $47, it obviously has no intrinsic value.

2. An option may have **no** intrinsic value today, but because of what might happen to the price of the underlying stock tomorrow, its premium will include, or consist solely of, a time value increment up until the day it expires.

   How much time value? Clearly the DEC 50 call from Table 1 is worth more to the speculator who owns it than the DEC 70 call on the same stock. But why 15.25 for the premium reflecting 13.37 intrinsic value and 1.87 time value vs. the straight time value of 3.50 on the higher strike?

		CALLS Exercise ("Strike") Price:			PUTS Exercise ("Strike") Price:		
Expiry Month	Months until Expiry	50	60	70	50	60	70
SEP	1	14.13	5.88	1.19	0.06	1.81	7.63
DEC	4	15.25	8.00	3.50	0.63	3.25	9.38
MAR	7	n/a	9.75	5.63	1.00	4.25	8.88

The Option Class Price/Time Matrix
(underlying stock at $63.38)

Table 1

An Irresistable Digression

It would be difficult to imagine a computer modelling project more attractive — and more prone to misunderstanding, misconstrual, misinformation and just plain mistakenness — than option price modelling and projection. Yet, measured by the depth in feet or metres of published opinion — approximately up to an aardvark's armpit — one could hypothesize that this single area of option/computer interface outstacks all other option-related CPU and Connect combined. Those among you whose interest may, however, now be aroused should be forewarned before, in Pooh-like disdain for caution, you arouse the bees. Fortunately, fair value modelling is not my subject today.

My subject today **is** option dynamics, and it should already be clear to you that although this brief delineation makes no claim to be any more expansive than required, it has sufficed to suggest that the buying and selling of puts and calls over time, as the matrix of series shifts in response to the price movement of the underlying stock, is replete with sufficient opportunities for tactical and strategic gamesmanship to sate even the more brilliant and ambitious gamblers and gamesters among us.

II. "Static" Option Tactics and Strategies

Simple Tactics: Opening Buys

The most straightforward, and best known — if not notorious — use of options is the simple opening buy of an option, especially a call. (An opening transaction initiates a position, a closing transaction cancels it out.) The opening buyer is usually hoping that the value of his purchase will rise sharply, in leveraged response to an appropriate move by the underlying stock, enabling him to close his position at a higher profitable price.

The purchase of options as a stand-alone tactic is a dangerous business, figuratively akin to the lone jeep sent deep into enemy territory with instructions to proceed until blown up, and then report back. In cold statistical terms, we glean from the 1979 Statistical Summary published by the Chicago Board Options Exchange the interesting fact that during that year, 23% of all opening call purchases expired worthless, representing a total loss of capital by their hapless owners.

Interestingly enough, the figure for the opening purchase of puts is almost as alarming, despite the vaguely-held conventional impression that bears, being pessimistic, are smarter than bulls. The simple fact is that purchasing either type of option as a stand-alone transaction is a risky business, involving an attempt not only to perceive the future **direction** of the underlying stock's movement, but its timing as well. The buyers of listed options have purchased wasting assets, which too often waste away... to nothing.

Yet they persist, and we have no intention of denigrating the purchase of options per se by speculatively-minded participants. The opening buy can hardly be characterized as intrinsically foolish or inappropriate. It is, to the contrary, often wisdom and appropriateness exemplified... especially from the retrospective viewpoint devolving from a subsequent closing sale at a sharply higher price. The foolishness enters, however, when a speculator deludes himself into characterizing his opening buy as an investment or considers his position to be anything other than a volatile, high-risk wasting asset.

We would draw your attention to Table 2, which lays out a complete set of puts and calls relating to a single underlying stock, each data line showing the prices at the close of trading on the last day of eight successive months.

In Table 3, we have created index numbers from the data of Table 2, whereby the percentage movement of the common stock and each series of options can be tracked from its first availability through August, 1980.

Finally, in Table 4, we have displayed the month-to-month closing price change as a percentage change over the 28, 30 or 31 day period.

Tables 2, 3 and 4 highlight, I believe, with increasing emphasis, the risks and rewards awaiting the opening buyer of listed puts and calls. The temptations are mouthwatering. The penalties for poor selection and timing, frequently devastating.

OPTION STOCK AND ASSOCIATED OPTION SERIES
MONTHLY CLOSING PRICES

		CALLS								
	COMMON	SEP 50	SEP 60	SEP 70	DEC 50	DEC 60	DEC 70	MAR 50	MAR 60	MAR 70
JAN/80	49.50	7.25								
FEB/80	58.38	13.75	7.75							
MAR/80	65.50	19.63	12.25	7.63						
APR/80	53.13	8.75	4.38	1.94	11.88	6.88	3.50			
MAY/80	57.00	11.00	6.25	2.50	14.25	8.00	4.38			
JUN/80	59.88	12.00	5.13	1.50	15.00	7.00	3.38			
JUL/80	62.88	14.25	6.25	1.63	15.00	8.38	3.50		9.50	5.50
AUG/80	63.38	14.13	5.88	1.19	15.25	8.00	3.50		9.75	5.63

		PUTS								
	COMMON	SEP 50	SEP 60	SEP 70	DEC 50	DEC 60	DEC 70	MAR 50	MAR 60	MAR 70
JAN/80	49.50	4.25								
FEB/80	58.38	2.25	5.50							
MAR/80	65.50	0.94	4.00	8.50						
APR/80	53.13	4.63	10.00		5.00	11.75				
MAY/80	57.00	2.25	6.00	13.00	3.13	7.00				
JUN/80	59.88	0.69	4.00	10.00	1.50	4.50	10.50			
JUL/80	62.88	0.38	2.88	8.50	1.06	4.13	9.25	1.88	5.25	
AUG/80	63.38	0.06	1.81	7.63	0.63	3.25	9.38	1.00	4.25	8.88

Table 2

OPTION STOCK AND ASSOCIATED OPTION SERIES
FIRST MONTHLY CLOSING PRICE INDEX = 100

		CALLS								
	COMMON	SEP 50	SEP 60	SEP 70	DEC 50	DEC 60	DEC 70	MAR 50	MAR 60	MAR 70
JAN/80	100.00	100.00								
FEB/80	117.93	189.66	100.00							
MAR/80	132.32	270.69	158.06	100.00						
APR/80	107.32	120.69	56.45	25.40	100.00	100.00	100.00			
MAY/80	115.15	151.72	80.65	32.79	120.00	116.36	125.00			
JUN/80	120.96	165.52	66.13	19.67	126.32	101.82	96.43			
JUL/80	127.02	196.55	80.65	21.31	126.32	121.82	100.00		100.00	100.00
AUG/80	128.03	194.83	75.81	15.57	128.42	116.36	100.00		102.63	102.27

		PUTS								
	COMMON	SEP 50	SEP 60	SEP 70	DEC 50	DEC 60	DEC 70	MAR 50	MAR 60	MAR 70
JAN/80	100.00	100.00								
FEB/80	117.93	52.94	100.00							
MAR/80	132.32	22.05	72.73	100.00						
APR/80	107.32	108.82	181.82		100.00	100.00				
MAY/80	115.15	52.94	109.09	152.94	62.50	59.57				
JUN/80	120.96	16.16	72.73	117.65	30.00	38.30	100.00			
JUL/80	127.02	8.82	52.27	100.00	21.24	35.11	88.10	100.00	100.00	
AUG/80	128.03	1.46	32.95	89.71	12.50	27.66	89.29	53.33	80.95	100

Table 3

OPTION STOCK AND ASSOCIATED OPTION SERIES
MONTH-TO-MONTH CLOSING PRICE PERCENT CHANGE

		CALLS								
	COMMON	SEP 50	SEP 60	SEP 70	DEC 50	DEC 60	DEC 70	MAR 50	MAR 60	MAR 70
JAN/80										
FEB/80	17.93	89.66								
MAR/80	12.21	42.73	58.06							
APR/80	-18.89	-55.41	-64.29	-74.60						
MAY/80	7.29	25.71	42.86	29.07	20.00	16.36	25.00			
JUN/80	5.04	9.09	-18.00	-40.00	5.26	-12.50	-22.86			
JUL/80	5.01	18.75	21.95	8.33		19.64	3.70			
AUG/80	0.80	-0.88	-6.00	-26.95	1.67	-4.48			2.63	2.27

		PUTS								
	COMMON	SEP 50	SEP 60	SEP 70	DEC 50	DEC 60	DEC 70	MAR 50	MAR 60	MAR 70
JAN/80										
FEB/80	17.93	-47.06								
MAR/80	12.21	-58.36	-27.27							
APR/80	-18.89	393.60	150.00							
MAY/80	7.29	-51.35	-40.00		-37.50	-40.43				
JUN/80	5.04	-69.47	-33.33	-23.08	-52.00	-35.71				
JUL/80	5.01	-45.41	-28.12	-15.00	-29.20	-8.33	-11.90			
AUG/80	0.80	-83.47	-36.97	-10.29	-41.15	-21.21	1.35	-46.67	-19.05	

Table 4

Simple Tactics: Opening Sells

The opening sale of **calls**, without owning or having access to the stock which the transaction potentially obligates one to deliver, is known as naked or uncovered call writing, depending upon the gentility of the circles one moves in. The opening sale of **puts** without owning an offsetting entitlement to "put" stock to someone else at a different exercise price is known as naked or uncovered put writing.

Naked writing of puts and calls results in a condition as vulnerable as it sounds. Unlike the opening **buyer** of a put or call, who can lose no more than his initial investment, the naked writer can find himself at the mercy of the market in the underlying stock, scrambling to purchase stock at a higher price than his exercise price, because he has (involuntarily) been exercised and must come up with the shares; or ending up the proud owner of stock purchased, likewise through an involuntary exercise, and often at a strike price well above the current market value.

For these reasons, naked writing is suitable for a very small percentage of market participants, who are subject to stringent financial and suitability requirements imposed and enforced by exchanges and brokerage firms.

Simple Tactics: Closing Sales and Closing Buys

The opening **buyer** of a put or call can dispose of his option through one of three circumstances:

- through a closing sell, wherein the option may be passed along to a new opening buyer,
- through exercise, wherein he or she purchases the underlying stock, in the case of call buyers, or sells the underlying stock in the case of put buyers,
- by doing nothing, in the case of an option with no intrinsic or time value remaining, and watch it expire, worthless.

The opening **seller** of a put or call can dispose of his or her position in one of three fashions, the first of which is involuntary:

- through an exercise "against", wherein the call seller is called upon to deliver stock, or the put seller is called upon to take delivery of the underlying stock,
- the initiation of a closing buy transaction, wherein the outstanding obligation is repurchased and thereby offset and cancelled,
- by doing nothing, in the case of an option with no intrinsic or time value left, and watching — with pleasure — as the contract expires worthless, cancelling his obligation and whetting his appetite to do it again.

The point we would emphasize here is that unless one chooses to exercise an opening buy, or is exercised against on an opening sell, and until expiry transpires, a position can be closed out through an offsetting transaction:

Opening Buy - Closing Sell
Opening Sell - Closing Buy

which takes place, of course, at the current market price of the option series involved. Thus the entire tactical arsenal of all players, excluding the exercise and expiry situations just listed, consists of the opening and closing purchase and sale of puts and calls. However, with the range of exercise prices and expiry dates available, we are confronted not with a paucity but an apparent surfeit of opportunities, to which we shall now turn.

Static Strategies

Since the earliest weeks of listed option trading, mathematically inclined participants, advisors, promoters, educators, have rushed to print with chapter and verse delineation of a wide variety of compound tactical opening configurations. Representing the deployment of two or more tactical components, such activities are clearly entitled to the appellation "strategy".

Clearly, the possibilities for combining opening buys and opening sells of puts and calls at various exercise prices and of varying expiration dates are enough to send an Integer Programming Model in one side of an Amdahl V8 and out the other, in a shower of bits and bytes.

Graphical depiction of a baker's dozen, showing the net resultant performance profile at expiry day, and over a range of possible underlying security prices at that time, is shown in Figure 1.

For some reason, the creativity evinced by the assembly of opening buy and sell components in diverse configurations does not extend, for the most part, to the nomenclature — which speaks of spreads, straddles and combinations. Exceptions include the Butterfly Spread, which is indeed comprised of a body constituted by two short options at an intermediate strike, and appended long option wings at higher and lower exercise prices on either side.

Much can be learned from playing with diverse configurations, assembling and dissembling opening buys and sells at various strike prices like so much stock market Lego. But before various among you head for the Exit, bent on advancing your retirement date through the discovery and deployment of an opening Grand Strategy, forgive me for dampening your enthusiasm by asserting that most of the time such plans don't work.

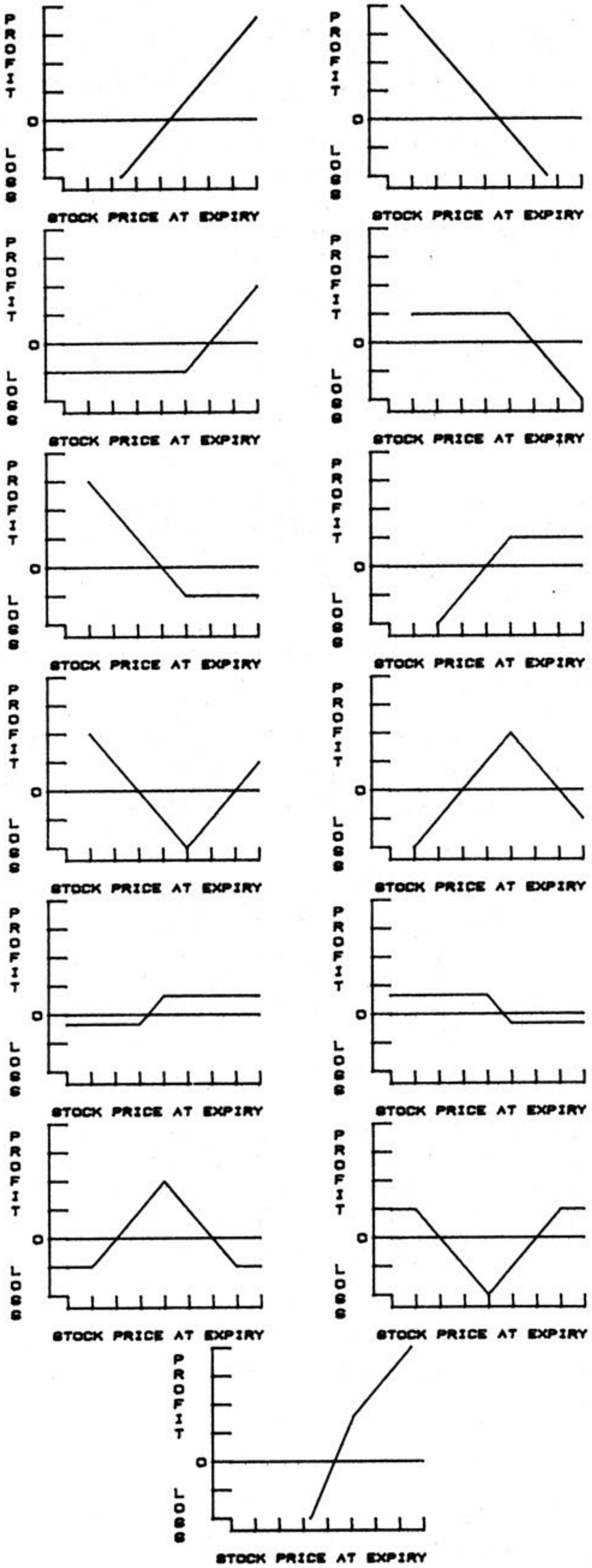

PROFIT
0
LOSS
STOCK PRICE AT EXPIRY
PROFIT
0
LOSS
STOCK PRICE AT EXPIRY
PROFIT
0
LOSS
STOCK PRICE AT EXPIRY
PROFIT
0
LOSS
STOCK PRICE AT EXPIRY
PROFIT
0
LOSS
STOCK PRICE AT EXPIRY
PROFIT
0
LOSS
STOCK PRICE AT EXPIRY
PROFIT
0
LOSS
STOCK PRICE AT EXPIRY
PROFIT
0
LOSS
STOCK PRICE AT EXPIRY
PROFIT
0
LOSS
STOCK PRICE AT EXPIRY
PROFIT
0
LOSS
STOCK PRICE AT EXPIRY
PROFIT
0
LOSS
STOCK PRICE AT EXPIRY
PROFIT
0
LOSS
STOCK PRICE AT EXPIRY
PROFIT
0
LOSS
STOCK PRICE AT EXPIRY

Figure 1

Option Market Efficiency

The efficiency of the options markets in adjusting the interrelated values of huge sets of alternative option series is invariably a source of amazement to those who delve deeply. With regular and dependable monotony, comparative option prices culled from the financial section of the daily newspaper turn out to be no more than rough indications of where the market was — not where it will be when the facilitating trade tickets submitted by your broker reach the floor.

In many cases, the attractive spread between two or more premiums may never have existed; the "last trades" may have occurred many minutes apart. If they did in fact exist at the same moment in time, a professional who operates from the floor of the exchange undoubtedly corrected the divergence from rational pricing in seconds, operating with one of the most sophisticated feedback-loop systems ever devised in the pursuit of self-regulating efficiency.

Small consolation though it may be, I can assure you that stockbrokers, aided by their electronic quote machines, fare only marginally better, in that their quotations are seldom instantaneous, and in heavy or active markets may be minutes or even more than an hour late. With the single exception of covered writing — the joint purchase of underlying stock and opening sale of a related option — most sophisticated option brokers will show a certain reluctance tn even attempt any but the most promising, straightforward compound opening trade.

If a final drawback were needed, brokerage commissions — the cost of doing business, or systemic "friction", depending on one's point of view — will often provide it, watering down or dissolving entirely what looked to be an immediate locked-in opening strategy.

Thus the major divergence between attractive paper-trading appearances and do-able compound opening positions can only be regarded as the shakiest of hopes for individual participants and a brokerage industry bent on grappling with and mastering the tactical and strategic promise options continue to hold out.

III. Where the Real Potential Lies

If the severely limited practicality of "simulataneous assembly" compound positions constituted the end-of-the-line for the evolutionary development of listed option use, the investment community would not be remiss in questioning the long-term viability of the option market, or at least its "market share" as a competitive alternative to the various modes of investment and speculation available to public and institutional participants.

However, it is far too early to ring down any rhetorical curtains. If the "simultaneous assembly" compound position is waning, the "dynamic" or staged-assembly compound position is nascent and rising.

Dynamic Strategies

Listed option strategies, if implemented in steps, compounded from tactical trades made in response to the ebb and flow of the price-action battle, can bring undreamed-of flexibility to the investor or speculator who can keep a cool head as market hostilities rage — consolidating paper profits when a turn of events provides them — partially or completely offsetting paper losses when his defences are breached — in short, playing the compleat opportunist.

Unlike the static strategy, which often relies on market inefficiencies and relationships which never were for its apparent existence, the dynamic strategy can exist and flourish in a fully efficient, real-world context.

Within the constraints imposed by the current forum, we can only proffer one straightforward illustration to document these broad assertions. Those who are sufficiently keen will have no difficulty in finding many, many more.

An Example

Consider the opening buyer of a call option, whose trepidation is overcome by the conviction that the underlying option stock is bound for glory.

In many cases, of which this is one, calls and puts **do** appreciate substantially in value at some point in their life — witness the price moves of Tables 2, 3 and 4 — but the too-human reluctance of the buyer to part with a winning position leads him to watch his paper profit melt away, to somehow vanish, through inaction which hindsight exposes as unforgivable. (Where is hindsight when we really need it, **before** the fact?)

Our option buyer opens his campaign with the following opening purchase:

Date	Transaction	Qty	Option	Series	Premium	Total Cost
21 Jul 80	Opening Buy	10	XYZ	Oct 22½	Calls at 2.625	2741.25

By August 8th, the underlying stock had risen 5 percent, from 23½ to 24¾. The option, however, had risen from the original purchase price of 2.625 to 4.00, a gain of 52 percent.

Taking commissions into account, we calculate that if the buyer had decided to take his paper profits on August 8th, his net gain would have been $1129, or 41%.

However, reluctant to give up his stake in a strong-performing stock, and aware of the tactical possibilities which have accompanied the rise in the common, he realizes that the XYZ Oct 27½ Calls (which were trading at .50 on July 21) are now trading at $1.70.

He resolves to work through the parameters of a bull spread, which would involve his

Opening Sale of 10 XYZ Oct 27½ Calls at 1.70.

This opening sale is covered, **not** naked, since although he is agreeing to potentially deliver stock at 27½ which he does not currently own, he is protected by the option he himself owns which entitles him to purchase the same security at 22½, if necessary.

By doing this opening sell, our speculator accomplishes the following:

- he creates a premium cashflow, after paying commissions, of $1598.00
- Since $1598 is 58% of his original investment, he has reduced his risk in the marketplace by that amount. He is no longer a candidate for the "total loss of capital" sweepstakes.

- The new capital amount at **risk** is $1143. His new potential **reward** amount would be based on an "ultimate" situation where the underlying stock would close in excess of 27½ on the October expiration date. Thus he would exercise his 10 October 22½ options, i.e., buy stock at that price, in order to deliver it to satisfy the exercise "against" him, i.e., sell stock, at 27½. Net of all commissions, his profit would be $3046, or 266% on his $1143 risk capital.

- If the stock closes under 27½ on expiration date, the 27½ option would expire worthless. In those circumstances, our speculator would simply sell his Oct 22½ options for their intrinsic value. Since his net investment is $1143, each option on would have to be worth $114, or approximately 1¼ points after commission for him to break even, i.e., somewhere around $24 per share, or somewhat less than where the common is currently trading.

While this tactical sequence is somewhat more complex than simply hanging on to a put or call option for dear life, it increases the chances of a profitable — or less than disastrous — outcome to a marked degree. It is more complex in the same way that chess is more complex than snakes-and-ladders; most of us gave up the latter in our childhood, just as today's astute investors understand the odds against making, and keeping, substantial stock market profits in a "simple", or "easy" manner.

In Optimistic Conclusion

The potential benefits which might accrue to investors and speculators through the use of dynamic, on-going tactics and strategies have been recognized, and used, by small numbers of astute market participants since early 1973. Seven years later, however, these approaches to the options markets have yet to gain wide currency.

The reasons are several. To begin with, widespread dissemination can only proceed as fast as the investing and speculating public are prepared to proceed. "Options" have entered the awareness of most investors, and sophistication is growing, largely due to the education and promotion undertaken by brokerage firms.

However, the dynamic use of listed options on anything but the most modest scale introduces two new requirements for brokerage firms which would do business in the new way:

- the ability to **monitor** client positions on a day-to-day basis, tracking performance not only of extant opening buy and opening sell positions, but, in addition, the performance of the associated series of puts and calls which are following every jog of the underlying stock according to their own parameters — thereby developing divergences which can often be turned to profit,

- the willingness to bring massive computer power to bear on the task of detecting, evaluating and documenting the range of dynamic moves which lie latent until exploited.

This is no easy task, not without risk. Such a program of client support involves a host of business decisions taken under uncertainty. Yet I have no doubt that it will be done, with the fruits of marketing success going, as usual, to those firms who are there first, with the best.

(The opinions expressed herein are solely those of the author, and do not necessarily reflect the policy or stance of Merrill Lynch Royal Securities Limited on the subject matter addressed.)

APL IN A LIBERAL ARTS COLLEGE

Donald B. McIntyre
Geology Department
Pomona College
Claremont, California

Introduction

Iverson notation was introduced at Pomona College in 1964 when the "Formal Description of System/360" was published in the IBM "Systems Journal", Vol. 3, #2-3. Pomona, a liberal arts undergraduate college with 1300 students, ordered a 360/40 on April 7, 1964, and one of the first System 360's shipped by IBM was delivered in September 1965. It had only 16K memory, and it had no console typewriter. The modern concept of a "System" was not recognized; all programming was in Basic Assembler (which required the card deck to be passed twice through the reader/punch), and the supposedly privileged instructions (such as LPSW) were freely available to any programmer.

The long-term advantage to me of so primitive and small a system was that it prepared me for the "Formal Description". If I had had a fully operational system, I would doubtless never have seen the IBM "Systems Journal", and would have remained in ignorance of Iverson's work. It was also fortunate that Chapter 2 of Iverson's "A Programming Language" used the IBM 7090 as the example of microcoding, because I was familiar with that machine through access to Western Data Processing Center on the U.C.L.A. campus. When I discovered the "Formal Description", and through it Iverson's book, I became greatly interested in the fact that a language had been developed that permitted the description of so complex a machine as a 360 computer. The intellectual potential of such a tool struck me as enormous and completely in the tradition of the historical development of mathematics. Quite apart from its obvious practical value, I was persuaded of its vital importance to the subject matter that is the core of liberal arts studies.

Although I had taught informal classes in Assembler and FORTRAN, I taught the first class for credit in computer science at Pomona in the academic year 1968-69. A feature of the class was a study of the comparative anatomy of computers, with special emphasis on the architecture of System/360 and its relationship to the structure of the IBM machines that preceded it. Iverson notation was used in the class, and students were encouraged to write formal descriptions of machines that they designed themselves. While I was teaching this class, I discovered to my delight that Iverson notation had been implemented as the language called APL.

Because no terminals were attached to our 360, we had to be content to use APL as a tool of thought, and this turned out to be very much to my advantage in appreciating its value. However, through the courtesy of our local IBM representatives, I was able to make some use of APL at the Thomas J. Watson Research Center, Yorktown

Heights, by means of terminals in IBM's Riverside office, California. On weekends and at night, some of my students were able to recast their Iverson notation into APL that they could execute on the computer at Yorktown Heights. The success of the class led me to attend the APL Users Conference ("The March on Armonk") at S.U.N.Y. Binghamton in July 1969, and I was fortunate also in being able to visit the APL group at Yorktown Heights.

From that time I worked to achieve an APL system for Pomona College, but this was not easy to accomplish. Although the configuration of our 360 grew, the administrative work on it also increased. It was never possible to attach terminals, and for many years we had access to APL only through the kindness of friends, until the College was able to lease a small number of ports on a DEC-10 owned by the other colleges in the Claremont cluster. During these years I did not teach the "Introduction to Computing" again, but on one occasion the College was fortunate in having Dr. W.J. Bergquist, of IBM, teach the course. In the meantime, I continued to use APL in my own work, and notably in my class in elementary crystallography, which is taken by all students who major in geology.

I was privileged also to spend the summer of 1971, at the invitation of Dr. Iverson, with the APL group, which was then under the management of Adin Falkoff at the IBM Scientific Center in Philadelphia.

In 1975, the Geology Department acquired the second 5100 desk-top computer shipped to a customer by IBM. Equipped with 4 video-monitors, the 5100 let me use APL in the classroom. Enormous progress resulted from the fact that students could get free use of APL 24 hours a day, and several developed into highly skilled users. Moreover other departments began to use the 5100 in Geology, and this machine was followed by a second 5100, in the Physics Department, and later by 5110's in Botany and Mathematics. All of these were used almost entirely as APL machines.

In 1977-78, through a grant from the Mellon Foundation for Faculty Development, and with the cooperation of the IBM Corporation, the College was able to have Dr. Don L. Orth teach courses both for faculty and students. This did much to lay the foundation at Pomona for a major advance in the appreciation of APL as an extremely important tool of thought.

On January 30, 1979, Pomona College ordered an IBM 4341, and an IBM 4331 was installed as an interim processor in May, 1979. This was the second 4300 System shipped to a customer. By this time it was generally understood on the campus that a good implementation of APL was essential, and this played a major role in our decision to purchase IBM equipment. During the first semester of 1979-80 we had limited access to VS APL under CMS through IBM 3277 terminals, which have since been replaced by 3278's. The Introductory Chemistry class, with about 150 students (which is a very large number for any class in Claremont), and some smaller classes in other departments, immediately began using APL on the 4331. During the second semester the APL Extended Editor was installed and 30 terminals were available to users. The number of active users rose to about 500. In addition to servicing the academic users, the 4331 continues to do most of the administrative computing for all the Claremont colleges formerly done by the 360/40. The administrative work is run as a virtual machine under the unsupported DOS-26 operating system, and there are, of course, times when the machine is overloaded.

Pomona College is a small liberal arts college with an exceptionally diverse curriculum, including Astronomy, Botany, and Geology. There are, however, no departments of engineering or computer science. Students who wish to do so are able to enroll in courses at our close neighbor, Harvey Mudd College, an undergraduate college strong in science and engineering. Harvey Mudd offers courses in computer science, but these have emphasized structured programming, notably ALGOL and PASCAL, in contrast to APL.

Because I had successfully demonstrated to colleagues in disciplines ranging from Economics to Chemistry that many practical problems can be solved remarkably easily be means of APL, I was asked to teach an "Introduction to Computing" in the second semester of 1979-80. The enrollment was about 140, making it one of the largest classes on the campus. Twenty-five different majors were represented by the students enrolled, and faculty members from such diverse areas as Biology, Geology, Art, French, and Physical Education audited the class. The impact of this class was sufficient to demand a two-week summer class in APL for faculty members, and it is probably safe to say that APL, using Direct Definition, is recognized as the principal language for computing at Pomona College.

The original System 360/40 was given to Pomona by Mr. Frank Roger Seaver of Los Angeles, and the further generosity of the Foundation he created made it possible for the College to purchase the 5100 computers, the 4331 (soon to be replaced by the 4341), and all the peripheral equipment. Without this support, it would, of course, have been impossible for Pomona to provide the equipment needed for quality instruction.

"Introduction to Computing": A Semester Course

I offered the course in 1979-80 to all interested students, without pre-requisites of any kind. The number of senior students who enrolled would have made a class of substantial size at Pomona, but had the class been restricted to seniors, no progress would have been made towards alleviating the problem in future years. Now that all students have had an opportunity to take the class, I intend to restrict enrollment and give preference to freshmen and sophomores.

Including auditors, nearly 160 people took the class, and the size required our meeting in a large lecture room in which it was impossible to use video-monitors. Had the room been smaller I would have used monitors attached to a 5100, because we lack the equipment needed to display the screen of a 3278 terminal to a group. The class met for two 80 minute periods each week. In addition to teaching this class, I taught classes in geology and performed administrative duties for the Geology Department and the Computer Center as best I could.

The concept of IBM 3278 terminals is to provide clusters that share a printer, and the largest of our clusters consists of 11 terminals in a rather small room in the Mathematics Department. I could not schedule the use of this room, and my students had to compete with others for the use of the terminals. I had the help of only one assistant, Jeff Siegel, who was a senior premedical student and greatly interested in the power and proper use of APL. At night he monitored the cluster of 11 terminals, giving help to any students who needed it, while I travelled round the other clusters, like a doctor making the rounds of the wards. In the future I hope to be able to schedule the students' time on the terminals, in the way that laboratory work is scheduled in the Chemistry Department. I would then restrict enrollment based on the number of laboratory sessions that can be staffed and the number of terminals available. Students who have already taken the course should be able to act as assistants in the future.

Because there were some people on campus who were not persuaded of the utility of APL, I considered the course to be, as its title indicated, an Introduction to Computing in general, and not merely an introduction to APL. My object was to give the students an understanding of the principles of computers and computer languages, and their historical development, so that every one could judge independently about the proper role and usefulness of APL. It would, of course, have been impossible for me to conceal my own judgement on this, and I did not pretend to do so.

Because we were using VM, every student had his own "Virtual Machine". The Conversational Monitor System (CMS) permits each virtual machine to have a "Profile" that is executed whenever the user logs on, and it would have been possible to have given each student a Profile that would have automatically invoked the APL system, but I chose not to do so. This meant that the students became familiar with the management of their resources under CP (Control Program) and CMS. For example they learned how to define and attach additional (temporary) disks; how to link to virtual disks of other users; how to write CMS EXECs (including their own Profiles); how to use IBM's full-screen editor (EDGAR); how to rename, copy, pack, and erase CMS files; and how to send CMS files to other users or to the line printer. It is easy to teach this kind of information when the student has a terminal in front of him, but I had to teach by remote control, and to do this I wrote a manual with the following sections:

1. CP CMS and APL: Three levels of operation

2. Interrupting your Virtual Machine

3. QUERY for information about your Virtual Machine

4. Copying CMS Files from disks on other Virtual Machines

5. Full-screen Editing and IBM's EDGAR editor

6. FLIST and the management of your resources under CMS

7. APL System Commands

8. IPF: IBM's full-screen Interactive Productivity Facility

9. Receiving and sending Mail

10. BROWSE

11. Reading and executing FORTRAN programs

12. Writing CMS EXECs

13. Attaching additional disk drives to your Virtual Machine

14. Linking to virtual disks on other machines

15. Using Program Function keys

16. Files created by the FORTGI compiler

17 FORTRAN subroutines

18. Management of APL Workspaces

19. APL Public Libraries

20. Direct Definition of APL functions

21. Relational Data Bases

22. QUERY: an APL simulation of IBM's "QUERY-BY-EXAMPLE"

23. Curve fitting (regression) and statistics using APL

24. Using Arrays in APL: an example from Economics

25. Logic: APL implementation of the work of Boole and Jevons

26. Linear Programming in APL

27. Introduction to plotting, using Tektronix 618 display and IBM's APL Graphpak

28. Data Bases with multiple-line entries, using APL

29. Storage of Floating-Point Numbers in the 4300

30. Introduction to Assembler Language

31. Introduction to the use of a memory dump

I asked each student in the class to attempt a project that would illustrate the application of computing to some area of personal interest. I particularly encouraged students to consult faculty members in their major fields, so that the projects would be as useful as possible in their academic programs. Several chose applications that related to their employment in college activities, such as the Student Union, the Admissions Office, the Development Office, and so on. A few concentrated on games. There were many projects dealing with statistics applied to psychology, government, and biology. Mathematics students developed functions for operations research and linear programming, and some created systems to differentiate and analyze given functions (see J.W. Bergquist, 1974). An interesting study of combinatorics made good use of recursive functions.

Many social scientists are highly dependent on a large black box called Statistical Package for the Social Sciences (SPSS), and several students successfully wrote APL functions that produce results identical to the output from SPSS.

The diversity was very great. Two students, impressed with the formal description of System/360, did a good job of simulating a 6800 microprocessor in APL. Another did an analysis of the practical question as to whether she should buy a new car, and if so, what kind. One biologist, who was using radio-transmitters on coyotes in the mountains, had the 5100 drive a Tektronix 4662 plotter to show on a contoured map the tracks of the animals by day and by night, and determine the statistics of their movements. Another analyzed tide tables to determine the length of time that the animals in the tide pools were below sea level. A French major used five-dimensional arrays to manipulate French verbs. And a geology student used eigenvalues to determine the fold axis in deformed rocks. Models in Economics were popular, and methods of text analysis were applied to English prose and to genetics.

After seeing a demonstration of IBM's "Query-by-Example", I simulated QBE in APL, largely in Direct Definition. I made the workspace available to the students, and in class I discussed how I had approached the problem. A large number of students took this as a model and created their own systems implementing simple but practical examples of relational data bases. An Art major prepared a system to catalog art slides, an anthropologist analyzed the distribution of cave art, a Music major simulated the R.I.L.M. bibliographic music data base, and an Asian Studies major created a complete interactive catalog of all known translations of Kabuki plays, which the faculty members in that field judge to be an important contribution.

Even those who did not complete a noteworthy project found the attempt to apply their computing knowledge to be a valuable experience.

Faculty Seminar in Computing

In June 1980, after the end of the second semester of the academic year, I taught a seminar for 12 of my colleagues, whose fields were Chemistry, Economics, Government, Mathematics, Philosophy, Sociology, and Zoology. The number of participants was limited to the number of terminals we could arrange to have in one room, for the reason that I wished each person to be at a terminal the entire time of the seminar. We worked intensively from 8:30 a.m. to 5:00 p.m. every day for two weeks. This was a very interesting experience. It involved total immersion in APL, and in my opinion the length of the seminar was just right. After 3-5 days most people find that they are becoming lost. They have learned much, but the parts are not yet seen as making a whole. As the second week progresses, it is obvious that comprehension (in the literal sense) is taken place rapidly; the mechanics of the operations cease to dominate, and attention can be given to topics of intellectual interest.

Although the group learned how to use the power of CMS, even learning how to manipulate real tapes, there was a clear emphasis on APL. Probably all who participated did so because they had heard of APL and wished to learn about it for themselves. Because they constantly learned by doing, they covered the more mechanical and machine-oriented material quickly, and this gave maximum time for instruction on the proper use of APL.

Many of the participants were from the Social Sciences, and I therefore covered two topics that I had not been able to include in the semester course: namely, the use of Shared Variables to access data in CMS files not created by APL, for example by EDGAR; and the use of IBM's 124X auxiliary processor for creating full-screens under the control of APL functions.

The faculty members who were in the seminar are busy people, and it seemed desirable to cover as much material as possible in the limited time, knowing that they could get the experience of practice later, on their own. I believed it was important that they ended the two weeks with the feeling that they had accomplished a lot, and that in dealing with students and colleagues in the future, they would be confident in knowing that although they might not remember exactly how to perform some task, they would remember that they had actually done it. In this way the fear and mystery associated with the jargon and the symbols was effectively dispelled.

Principles Used in APL Instruction

I always begin by showing that APL is immediately useful, for example by using `+/1 2 3 4 5 6` to add up a string of numbers. After two or three class meetings, when the students had had time to see how easily they could achieve results with APL, I was asked "when will we start writing programs?", or, in other words, "when will this start to hurt?" I decided then that it was worth taking time to let the students learn enough FORTRAN to write simple programs and subroutines. This ensures an understanding of the fact that there are different kinds of programming languages. Before I had finished with this exercise, several students asked me why we were wasting time on FORTRAN when it was so easy to achieve the same results in APL. I explained that there are those who claim that APL is inefficient in comparison to a compiled language, such as FORTRAN, and I directed them to colleagues who hold this view. The result was remarkable. The students asked whether the object was to save their effort or the effort of the machine! In view of the obvious decrease in cost in hardware, and the simultaneous increase in the cost of getting qualified people, the argument about supposed efficiency was not found to be very convincing. This was even more the case after we had discussed the "efficiency" of the FORTRAN program that we had written to do sorting. An important lesson learned was that inefficient programs can be written in any language, and that the efficiency of the interpreter or compiler might be taken into account also.

As the course continues I try constantly to make what we are doing appear obviously practical. I do this by taking examples that I know to be relevant to fields represented by members of the class. But I also emphasize the evolution of mathematical thought and notation through time, demonstrating that APL today is a continuation of the tradition represented by such men as Napier, Oughtred, Leibniz, Euler, Boole, Jevons, De Morgan, Cayley, Sylvester, Peano, and many more of the intellectual giants of past centuries. For example, we see how Iverson extended De Morgan's Law and why these are useful; how compression implements Boole's selection function; how the inner product of APL enormously extends the power of the matrix multiplication of Cayley; and how APL helps the non-mathematician to understand and use concepts such as zero, emptiness, and recursion.

Index Origin

The discovery of zero was one of the triumphs of human intellect. But, even today, many people, even mathematicians, seem not to have fully accepted zero into their thinking. Although my students were, of course, free to work in either origin, every expression that I shared with the class was written in Origin Zero. `X∘.*⍳N` is the obvious way to generate the powers of *X* for polynomials and regression, and the first (leading) power is `0`. The word "first" is, perhaps, the greatest stumbling block in the general acceptance of origin zero, but nearly all students chose to work in origin zero when they realized that graphs are normally drawn so that the axes are labelled starting at 0, and when they saw such commonly occurring expressions as:

`N+⍳M` to generate *M* integers starting with *N*

`A[B]` where *B* is logical (Boolean); i.e., 0's or 1's. This is an expression useful for giving pictorial output, as in

```
                                        ' ⋆'[⊖(ι⌈/V)∘.≤V←3 2 1 0 2 4 6]
```

```
(ρM)⊤(V=⌈/V)/ιρV←,M
```

which gives the indexes of all occurrences of the largest value in the matrix *M*

Right-to-Left Execution

Not a single student as much as remarked about the right-to-left order of execution. At the outset I simply said that, as in the expression log sin square-root X, every function in APL takes as its right argument everything to its right, unless parentheses change the order. I pointed out that this was the normal order in mathematics beyond the most elementary level, for example in a sequence of rotation matrices, and that it leads to expressions that use the minimum of parentheses. I stressed that it is never necessary to end an expression with a right parenthesis, or to use two adjacent right parentheses within any expression.

The non-commutative functions minus and divide, which may require parentheses if the left argument is composite, can often be converted into the related commutative functions plus and multiply, with the elimination of parentheses:

```
(A F B)-1     ↔     ¯1 + A F B
(A F B)÷10    ↔     .1 × A F B
```

Moreover, this brings out the analogy between the negative sign and the decimal point as part of the name of a number.

When these simplifications are ignored, there is ground for questioning whether the concepts of negative and fractional numbers have been understood.

Although expressions can be EXECUTED only from right-to-left, it is good to adopt the practice of READING expressions from left-to-right and well as from right-to-left. For this reason (as originally pointed out to me by Larry Breed) no variable should be specified more than once in one expression.

I asked my students to be conscious of the style of their APL, just as they should be of their English composition. As with English, it is important to study good examples. The publications of the APL Press were pointed out as containing admirable models.

Direct Definition

The word "function" was first used in the modern sense by Leibniz and Bernoulli before 1700. They wrote in Latin and simply used the word meaning "perform". A function performs the task of combining its arguments together in some specified (formal) way. If the arguments are thought of as nouns, then the function is an imperative verb.

In 1753, Euler used the notation Φ:(x,t) for a function Φ whose arguments are x and t, and in 1754 he wrote the analogous notation f:(a,n) for the function f of a and n.

K.E. Iverson, the principal inventor of APL, introduced a variant of Euler's notation in 1976 in his book "Elementary Analysis". Like Euler, Iverson began by writing the name of the function separated from what followed by a colon. Whereas Euler simply named the arguments (separated by commas and enclosed in parentheses), Iverson wrote the APL expression that constitutes the formal definition of the function. "A formal definition is one which can be interpreted by a mechanical application of known rules, requiring no judgement or subtle interpretation" (Iverson, 1976, p. 10).

Following the conventions of elementary mathematics (illustrated in the examples -*X*, 2+3, 2-3, 2×3, and 2÷3), Iverson permitted definition of functions with either one or two arguments, one on the right and, if necessary, one on the left of the function name. To distinguish between these within the formal definition, he used the symbol α to stand for the left argument (if any), and ω to stand for the right argument. These are "dummy" arguments or "place-holders", for which actual values are substituted when execution takes place.

Thus, we can define the function *YF* as follows:

```
      YF:(ω∘.*ιρα)+.×α
```

where α is a vector of length 2 whose elements are the y-intercept and the slope of a straight line, and where ω is a vector (or scalar) of values of x.

Then *YF* is the function that gives the values of y corresponding to the values of x denoted by ω; for example,

```
      .5 2 YF 1 2 3 4 5
```

gives the values of y corresponding to the values 1 2 3 4 5 for x when the y-intercept is .5 and the slope is 2. The colon may be read as "is"; thus, *F*:α*ω may be read as "*F IS* α raised to the power ω".

Iverson extended this notation to permit definition of recursive functions; i.e., functions that are defined in terms of themselves (Iverson, 1976, Chapter 10). The recursive definition of a factorial requires the following three pieces of information:

A primary expression:	ω×*FAC* ω-1
A proposition:	ω=0
A secondary expression:	1

In a formal definition, these three items are presented in order, with colons separating them; thus

```
      FAC:ω×FACω-1:ω=0:1

      FAC 4
24
```

You can read the formal definitions as: "The *FAC*torical *IS* ω times the factorial of ω-1, UNLESS ω=0, IN WHICH CASE it is 1".

A proposition is an expression that yields either 0 (meaning false) or 1 (meaning true). When there are three colons, the proposition is executed first; the primary expression is executed if the proposition yields 0, and the secondary expression is executed if the proposition yields 1.

Iverson's αω method of function definition is called "Direct Definition". In order to implement it on a computer, it is necessary to provide a compiler that can take the character string giving the formal definition and convert it into executable code. I.P. Sharp's system can detect function definition from its unique syntax (the name being followed by colon) and does the conversion in a manner transparent to the user.

Direct Definition enforces a discipline that makes it far easier to write good APL. It is also a natural way to teach, because functions are introduced (on the blackboard or on the terminal) as the subject requires them and without digressions. Any expression already developed is simply given a name and used. Documentation is straightforward, because every distinct concept is a one-line function with no side effects; i.e., without pretending to do one thing but doing another. If the functions have been well chosen and well named, it is easy to state what each does. Interactive documentation with appropriate data can illuminate the individual functions and illustrate typical ways in which the functions interact as a system.

As Paul Berry has said (Berry, et al., 1971), "When APL functions are executed at an APL system, the user should have to state mathematical kernel of his function, and nothing more". Direct Definition permits just that. Nothing superfluous is required. I believe that the use of Direct Definition is also consistent with the exhortation of John Backus, in his Turing Award Lecture, that the user should function as much as possible in the world of expressions rather than in the world of control statements.

Throughout both my semester course and my two-week seminar, I used only Direct Definition of functions. The only exceptions were when I needed "house-keeping" functions, for example to handle shared variables or to provide annotated output. Despite my example and admonition, several students "discovered" that the del (∇) form of function definition permitted them to write multi-line programs filled with loops and branches. Although I had never once spoken about branching, students who already knew something about languages such as BASIC seemed to have a serious disadvantage in being almost incapable of escaping from the intellectual tyranny of "word-at-a-time processing" and the consequent superstructure of control statements. In an attempt to combat this unfortunate tendency, I played the tape of John Backus' Turing Award Lecture, in which he says that this style of programming has held up progress for 20 years. My experience persuades me that the widespread use of BASIC on home-computers is a dangerous threat to the intellectual development of many bright people.

Familiarity with All Primary Functions

I try to let my students share the intellectual pleasure of seeing the surprising and diverse uses of such functions as ⊥ and ⊤. I want them to feel at home with expressions like 0⊥*V* (for the scalar representation of the last element of the numeric vector *V*), or *M*⊥1 (for 1 plus the number of 1s before the first 0, reading the rows of the logical matrix *M* from right to left). On the first encounter with such expressions, they may appear awkward and unnecessarily subtle, but this is because of unfamiliarity with the primary functions. Like any other new concept, once the strangeness of novelty wears away, these functions can become very natural as well as useful intellectual tools.

As an example of how the decode, or base-value, function (⊥) can be introduced and made familiar, consider the following:

It is easy to understand how the vector 1 9 8 4 is reduced to a scalar by the function ⊥ using the base 10

```
      10⊥1 9 8 4
1984
```

Now create a function that provides a formal definition of what we have just performed, and we see that although we commonly write a sequence of numbers from left to right, we write the digits of a single number so that they increase in value from right to left; thus, the vector resulting from the function ⍳ must be rotated before it can be used to generate the needed weights.

```
      BASE:ω+.×⌽α*⍳⍴ω

      V←1 9 8 4

      10 BASE V
1984
```

The function *BASE* will always give a scalar result, and its left argument can be 0:

```
      0 BASE V
4
      0⊥1 9 8 4
4
```

That the result is a scalar can best be demonstrated by:

```
      ⍴⍴0⊥V
0
```

In contrast, the result of take (↑) is always a vector:

```
      ⍴⍴¯1↑V
1
```

Now consider the treatment of a polynomial such as:

$$a + bx + cx^2 + dx^3$$

In the first place we can rewrite this as:

$$a_0x^0 + a_1x^1 + a_2x^2 + a_3x^3$$

so that the pattern is more evident by having a uniform and consistent treatment of all terms. We have made it obvious that there are only two arguments: a vector of coefficients and a scalar, *X*. The first term is brought into conformity with the pattern of the others, once we assimilate zero into our thinking. Moreover, we find that many disciplines, such as Economics, already use the subscript 0 for the first term. Thus origin zero is in fact in common use, though not always recognized. I encourage my students to be consistent in their use of origin zero.

If we take the following values for the coefficients and the scalar:

```
      C←1.5 2 ¯3 0.5
      X←2.5
```

the polynomial can be written in APL thus:

```
      C[0] + (C[1]×X) + (C[2]×X*2) + C[3]×X*3
¯4.4375
```

This is a direct translation of the algebra into APL. We can improve it by rewriting so that the parentheses are not needed:

```
      C[0]+X×C[1]+X×C[2]+X×C[3]
¯4.4375
```

But notice that although the coefficients exist as an array, we refer to them and use them, in both expressions, as individual scalars. In order to take advantage of the fact that C is an array, and that the vector of powers to which X is raised is itself a function of C, we write:

```
      +/C×X*⍳⍴C
¯4.4375
```

Now, whenever we see the expression +/A×B, we should consider whether this can be rewritten as A+.×B. In this case, as so often happens, we have a sum of products, and the necessary compatibility enables us to write:

```
      C+.×X*⍳⍴C
¯4.4375
```

We can therefore use the function BASE to achieve the same result:

```
      X BASE ⌽C
¯4.4375
```

Or, what is the same thing:

```
      X⊥⌽C
¯4.4375
```

It should be noted that because C is a vector (⍴⍴C ↔ 1), its leading axis is also its last, and hence ⌽C ↔ ⊖C.

We have looked at the case where the X of the polynomial is a scalar, but if X is a vector, then the polynomial must be evaluated for each value of X:

```
      POLY:(⍺∘.*⍳⍴⍵)+.×⍵

      X←2.5 3.5

      X POLY C
¯4.4375 ¯6.8125
```

If we use the base function to achieve the same result, X must be a matrix. If it were a vector, its elements would be used together (like 24 60 for hours and minutes) instead of separately.

```
      (X∘.+(⍴C)⍴0)⊥⌽C
¯4.4375 ¯6.8125
```

Or, as we will see is still better, (X∘.+(⍴C)⍴0)⊥⊖C

Now suppose that several polynomials are to be evaluated, each with its own set of coefficients:

```
      C←C,[.5]2 1.3 ¯1.5 .25
      C
1.5   2
2     1.3
¯3   ¯1.5
0.5   0.25
```

In conformity with the rules for matrix multiplication, where the rows of the left argument are combined with the columns of the right argument, the coefficients of the individual polynomials must be columns. Now that the *C* is a matrix, ⌽*C* would simply change the order in which the polynomials are given — rotating about the last axis. As long as *C* was a vector it had only one axis, but now there are two, and to change the order of the terms within each polynomial, we must rotate about the leading axis, ⊖*C*.

```
      (X∘.+(1↑⍴C)⍴0)⊥⊖C
¯4.4375  ¯0.21875
¯6.8125  ¯1.1063
```

Each polynomial is evaluated for each term in *X*.

To see how the *BASE* function must be modified to produce the same result, we can proceed as follows:

```
      BASE1:((1↓⌽×\⌽α),1)+.×ω
      ⍝  α⊥ω where α is vector
```

Or rearranging to eliminate unneeded parentheses:

```
      BASE2:(1↓⌽1,×\⌽α)+.×ω
```

```
      24 60 60 BASE2 1 2 3                  The number of seconds in
3723                                        1 hour, 2 minutes, and 3 seconds
```

If α is a matrix and ω is a scalar, vector, or matrix:

```
      BASE3:(0 1↓⌽1,×\⌽α)+.×ω
```

If *A* and *B* are specified as matrices, as follows:

```
      A
  24  60  60                       Hours, minutes, seconds
1760   3  12                       Yards, feet, inches

      B
1 4
2 5
3 6

      A BASE3 B
 3723 14706
   63   210

      A⊥B
 3723 14706
   63   210
```

The function *BASE*3 will work if α is a matrix, but not if α is a vector. To allow for this, we define:

```
      BASE4:((⌽(ρρα)↑1)↓⌽1,×\⌽α)+.×ω
```

Or the clumsier, but for some students more easily understood, form:

```
      BASE5:((((¯1+ρρα)ρ0),1)↓⌽1,×\⌽α)+.×ω

      24 60 60 BASE4 B
3723 14706
```

But now this will not work if α is a scalar. A simple way out of the difficulty is to decide which algorithm is appropriate by using the so-called "recursive" form, even although the function is not itself recursive:

```
      BASE6:αBASE4ω:0=ρρα:(α*⌽ι1↑ρω)+.×ω

      A BASE6 B
 3723 14706
   63   210

      10 BASE6 B
123 456
```

If both α and ω are scalars, then the result is ω (weighting factor of 1):

```
      BASE7:αBASE6ω:0∧.=(ρρα),ρρω:1×ω
```

After exploring the base function ⊥ in this way, the student should have no difficulty in understanding why 0⊥1 2 3 4 is the scalar 4, or why 1 0 1 1⊥1 ↔ 3. It should also be clear why the following function will right adjust a character matrix:

```
      RAJ:(1-(ω=' ')⊥1)⌽ω
```

and why the following will give the vector of indexes that will alphabetize a character matrix provided that ∘⊥ρω is not too large:

```
      ASRT:⍋(ρ⎕AV)⊥⍉⎕AVιω
```

Roy Sykes (1978) has given the following good example of ⊥.

```
      SPS:(,α)[(ρα)⊥(1⌽ιρρω)⍉ω]
      ⍝ Scattered-Point Selection from array α by indices ω
```

To illustrate the utility of ⊤ we might introduce a function that will find the indexes of all occurrences of ω in a matrix α:

```
      IXM:(ρα)⊤(V=ω)/ιρV←,α
```

Those who began by thinking that because base 10 had been good enough for their fathers it should be good enough for them, are often excited to discover that there are many ways in which ⊥ and ⊤ can accomplish very practical tasks. They are then ready to read the sections on Decode and Encode in Falkoff and Orth (1979, p. 435-436).

It was interesting that, after conducting this exploration in the faculty seminar, a zoologist realized that in his work with tide tables he converts matrices of times in days, hours, minutes into vectors of minutes, to learn how long the tidepools are submerged over a given period. He saw that he could get his result using the function ⊥ with the left argument `0 24 60`, and with the right argument a matrix whose rows are days, hours, minutes, and whose columns are points in time.

We have taken the decode function as an example, because it is often not well understood by beginners, but every primary function deserves similar study.

Using Arrays in APL

"It is the constant aim of the mathematician to reduce all his expressions to their lowest terms, to retrench every superfluous word and phrase, and to condense the Maximum of meaning into the Minimum of language." (1877)

"A matrix of quadrate form ... emerges ... in a glorified shape — as an organism composed of discrete parts, but having an essential and undivisible unity as a whole of its own. The conception of multiple quantity thus rises upon the field of vision. [Matrix] ... dropped its provisional mantle, its aspect as a mere schema, and stood revealed as bona-fide multiple quantity subject to all the affections and lending itself to all the operations of ordinary numerical quantity. ... The Apotheosis of Algebraical Quantity." (1884)

These quotations, taken from the writings of James Joseph Sylvester (1814-1897), the self-styled Mathematical Adam, who introduced the term Matrix in 1850, summarize for me much of the spirit of APL, and testify to the importance of APL in the development of mathematics.

> When APL is well written, it takes full advantage of the algebra of multiple quantity, and would surely have greatly pleased Sylvester. When FORTRAN employs a loop, it is likely that this can be avoided in APL by increasing the rank of a variable from a scalar to a vector. If FORTRAN has nested loops, APL probably increases the rank still further. Thus the concept of rank must be thoroughly grasped from the beginning, and the student should be trained to think about the `ρρ` of his variables.
>
> I have found it important to demonstrate how one approaches a problem by considering the shapes and compatibilities of the variables. For example, to create a histogram you must have two variables: the data to be classified and the bounds that define the classes. Because there is no necessary compatibility between these, an outer product is required:
>
> ```
> HIST:PWD +/α∘.>ω          Where α gives the bounds and ω the data
> PWD:(1↓ω)-¯1↓ω           Pairwise differences
> ```
>
> Any function written for scalars only should be considered as a candidate for extension so that it can work with vectors, and possibly with matrices and higher arrays. The person familiar with the subject matter should enquire what meaning is to be attached to such proposed extensions. Scalars are simply scalars, and compatibility between them is always guaranteed, but arrays can differ in both rank and shape.
>
> The following example is a function written by an economist as part of a system to compute Double Declining Balance. The formula is cumbersome, and the APL reflects this; but the important point is that the function as written works only with scalars.

```
      ∇ Z←R DDB1 T
[1]   Z←((2÷T)÷R+2÷T)×1-((1-2÷T)÷1+R)⋆⌈T÷2
      ∇
```

The following functions, in Direct Definition, take into account the necessary compatibilities of the arguments, and outer products are generated between the vectors of different lengths.

```
      DDB2:(((⍴M)⍴2÷⍵)÷M←α∘.+2÷⍵)×⍉αPVA⍵

      PVA:1-((1-2÷⍵)∘.÷1+α)⋆⌈.5×⍵∘.+(⍴α)⍴0
```
Present Value (part A)

Where compatibilities necessarily exist between the variables, then inner products are probably called for. Students should know that

if Z←A∘.×B then ⍴Z ↔ (⍴A),⍴B

but if Z←A+.×B then the condition for compatibility of arrays is that ¯1↑⍴A ↔ 1↑⍴B

and ⍴Z ↔ (¯1↓⍴A),1↓⍴B

An interesting example of extending a function that worked only on scalars arose in an ecological study. Brillouin's measure of diversity of species requires the logarithms of the factorials of large numbers. The direct APL expression for small numbers is:

```
      ⍟!5
4.7875
```

Because this fails for large numbers, the following recursive function was defined:

```
      LFA:(⍟⍵)+LFA⍵-1:⍵=0:0
      ⍝ Log of factorial for large scalar ⍵

      LFA 5
4.7875
```

But Brillouin's Index uses the sum of the logarithms of the factorials, and LFA only works with scalars, hence the functions:

```
      SLF:+/⍟(V≠0)/V←,M×+\M←⍵∘.>⍳+/⍵
      ⍝ Sum logs of factorials (for BDI)

      BDI:((⍟10)÷+/⍵)×(SLF+/⍵)-SLF⍵
      ⍝ Brillouin's Diversity Index

      SLF 5
4.7875

      SLF 4 2 5
8.6587

      +/⍟!4 2 5
8.6587
```

Taking the data from Robert W. Poole (1974, p. 390):

```
      V
235 218 192 87 20 11 11 8 7 4 3 2 2 1 1

      H←BDI V
      H
3.8063        Brillouin's measure is designated H

      H÷⍟10
1.653         natural bels per individual
```

the number of individuals is:

```
      +/V
802

      B←H×+/V
      B
1325.7        The total information content of the collection, in natural bels
```

The following example, proposed by an economics student, illustrates the use of arrays in defining functions for practical applications.

The economist Paul Douglas (later U.S. senator) wanted to relate the Quantity of output to the Capital (number of machines) and the Labor (number of workers). Together with a mathematical colleague, Cobb, he defined an expression that is known as the Cobb-Douglas Production function (Douglas, 1976):

$$Q = A.K^{\alpha} L^{\beta}$$

where *K* is the Capital (e.g., number of tractors),
L is the Labor (number of farm workers),
and *Q* is the Quantity produced (e.g., amount of wheat)

A is a "shift parameter", which enables us to change the value of *Q* without changing *K* or *L*.

α and β are numbers (usually between .25 and .75) that describe the way in which *Q* changes as *K* and *L* are changed. There is a special interest in the case where $\alpha + \beta = 1$; i.e., where $\beta = 1-\alpha$

If ω is the vector α, *A*, *K*, *L*, then *Q* is given by the function

```
CDP:ω[1]×(ω[2]⋆ω[0])×ω[3]⋆1-ω[0]
```

For example,

```
      CDP .3 10 2 1
12.311
```

However, when the economist writes the function as `Q=f(K,L)`, which can be read as "*Q* is a function of *K* and *L*", he shows that he thinks of *K* and *L* as being fundamentally different from α and *A*. We might therefore separate the "parameters" α and *A* as the left argument, and let *K* and *L* together form the right argument of the function.

```
CDPO:α[1]×(ω[0]*α[0])×ω[1]*1-α[0]

       .3 10 CDP0 2 1
12.311

       .5 15 CDP0 3 4
51.962
```

But this function permits only scalar values for *K* and *L*, and we would like to see the effect on *Q* of varying *K* and *L*. The function ought to accept vectors of *K* and *L*. Now for every value of *K* there must be an associated value of *L*; so the right argument must be a matrix. Let us assume that the columns are *K* and *L*. Then, instead of raising *K* and *L* separately to the appropriate powers, we use the dyadic * with matrix arguments:

```
CDP1:α[1]××/ω*(ρω)ρα[0],1-α[0]

       .3 10 CDP1 2 2 3 3,[.5]1 2 3 4
12.311 20 30 36.693

       .3 10 CDP1 2,[.5]1 2 3 4
12.311 20 26.564 32.49
```

We might modify this function so that it could accept a vector of values for *A*. One way to do this is to make the economists' parameter α a global variable, *A*, thus reserving the left argument for the "shift parameter" or "constant".

Whereas the left argument of the × was previously a scalar, and so was automatically extended to be compatible with the vector right argument of ×, the left argument is now a vector and incompatible in length with the right argument; consequently an outer product is needed:

```
CDP2:α∘.××/ω*(ρω)ρA̲

       A̲←.3 .7
       10 15 20 CDP2 2,[.5]1 2 3 4 5
 12.311 20      26.564 32.49  37.983
 18.467 30      39.846 48.735 56.974
 24.623 40      53.128 64.98  75.966
```

When we saw the expression (*A***B*)×(*C***D*) in the first form of the function, we knew we could make *A* and *C* into the *COLUMNS* of a matrix *M*, and *C* and *D* into the *COLUMNS* of a compatible matrix *N*. We then wrote ×/*M***N*. But this form, f/ M g N (where f and g are scalar dyadic functions, taking scalar arguments and extending to element-by-element functions on arrays), can always be rewritten as an inner product, M f.g N; thus we have:

```
CDP3:α∘.×ω×.*A̲

      A̲←.3 .7
      10 15 20 CDP3 2,[.5]1 2 3 4 5
 12.311 20       26.564 32.49  37.983
 18.467 30       39.846 48.735 56.974
 24.623 40       53.128 64.98  75.966
```

This has the advantage of automatically extending to a matrix of A̲, whose *COLUMNS* are successive pairs of exponents. It is worth noting that the function is unchanged if a THIRD variable is added to *K* and *L*.

```
      A̲←X,[¯.5]1-X←.25 .3 .5
      A̲
 0.25 0.3  0.5
 0.75 0.7  0.5

      R←10 15 CDP3 2,[.5]1 2 3 4 5
      ρR
2 5 3
```

The result is rank 3; i.e. it has three axes for indexing. The axes are:

0	Length 2	2 ↔ ρ10 15	the economists' parameter *A*
1	Length 5	5 ↔ ρ*K* ↔ ρ*L*	the economists' variables
2	Length 3	3 ↔ 0⊥ρA̲	the number of exponent pairs

It is important to be able to identify any element in *R*, *R*[*I*;*J*;*K*], in relation to the original data. The function *CDP* (which takes only scalar arguments) should be used to check your navigation through the 3-dimensional array *R*.

```
      R
 11.892 12.311 14.142
 20     20     20
 27.108 26.564 24.495
 33.636 32.49  28.284
 39.764 37.983 31.623

 17.838 18.467 21.213
 30     30     30
 40.662 39.846 36.742
 50.454 48.735 42.426
 59.645 56.974 47.434
```

The sequence of the axes is determined by the sequence of the inner and outer products in the function *CDP*3, but it would probably be more convenient to rearrange the axes so that axis 0 corresponded to the exponent pairs, axis 1 to the shift parameter *A*, and axis 2 to the successive pairs of values of *K* and *L*. This is done by the dyadic transpose. To understand this, consider that the transpose of a matrix is the result of interchanging the rows and columns; i.e. the result of taking axis 1 as the leading axis and axis 0 as the last axis.

The monadic transpose of a 2-dimensional array (i.e. of a matrix) is merely a special case of the general (dyadic) transpose of ANY array. However, because a vector has only one axis, *V* ↔ ⍉*V*, and a scalar has NO axes to be permuted.

```
       M ↔ 0 1 ⍉ M
      ⍉M ↔ 1 0 ⍉ M
```

Consequently, we can rearrange the axes of *R* in this way:

```
      Q←1 2 0⍉R
      ⍴Q
3 2 5

      Q
 11.892 20       27.108 33.636 39.764
 17.838 30       40.662 50.454 59.645

 12.311 20       26.564 32.49  37.983
 18.467 30       39.846 48.735 56.974

 14.142 20       24.495 28.284 31.623
 21.213 30       36.742 42.426 47.434
```

In order to explore the properties of the Cobb-Douglas Production function, it might be useful to provide more structure in the variable. This can be done by raising its rank from 2 to 3; for example:

```
      I←4 5 2⍴⍉1+0 5⊤⍳20

      A←10 15 20

      I
 1 1
 1 2
 1 3
 1 4
 1 5

 2 1
 2 2
 2 3
 2 4
 2 5

 3 1
 3 2
 3 3
 3 4
 3 5

 4 1
 4 2
 4 3
 4 4
 4 5
```

Then the product of the powers is:

```
      R←I×.*X,[¯.5]1-X←.3 .5
      ⍴R
4 5 2
```

Notice that if

```
      R←A+.×B
```

then

```
      ρR ↔ (¯1↓ρA),1↓ρB
```

The last axis of the first argument must equal the first axis of the second argument, and it is these two equal axes that are eliminated in the inner product.

The rank is raised to 4 when we use the outer product to multiply by the vector of shift parameters. The new leading axis has length 3, equal to ρA:

```
      Q←A∘.×R
      ρQ
3 4 5 2
```

Once again rearranging the axes, we have:

```
      Q←1 2 3 0⍉Q
      ρQ
2 3 4 5
```

We therefore have 2 sets of results, corresponding to the 2 sets of exponents; each set being divided into 3 groups, corresponding to the 3 values of the shift parameter; and each group being a 4 by 5 matrix, corresponding to that part of the structure of the variables in array I that is not lost in the inner product.

```
      Q
 10      16.245 21.577 26.39  30.852
 12.311 20      26.564 32.49  37.983
 13.904 22.587 30      36.693 42.896
 15.157 24.623 32.704 40      46.762

 15      24.368 32.365 39.585 46.278
 18.467 30      39.846 48.735 56.974
 20.856 33.88  45      55.039 64.344
 22.736 36.934 49.056 60      70.144


 20      32.49  43.153 52.78  61.703
 24.623 40      53.128 64.98  75.966
 27.808 45.174 60      73.385 85.792
 30.314 49.246 65.408 80      93.525


 10      14.142 17.321 20     22.361
 14.142 20      24.495 28.284 31.623
 17.321 24.495 30      34.641 38.73
 20      28.284 34.641 40     44.721

 15      21.213 25.981 30     33.541
 21.213 30      36.742 42.426 47.434
 25.981 36.742 45      51.962 58.095
 30      42.426 51.962 60     67.082
```

```
20     28.284 34.641 40     44.721
28.284 40     48.99  56.569 63.246
34.641 48.99  60     69.282 77.46
40     56.569 69.282 80     89.443
```

Gathering up the individual pieces, we have the following result:

```
      1 ↔ ∧/, Q = 1 2 3 0 ⍉ A∘.×I×.*A̲,[¯.5]1-A̲
```

Values of *Q* can be contoured on a plot of Capital, *K*, against Labour, *L*. Different contoured maps correspond to different values of the shift parameter, *A*, and α.

A perhaps simpler example of the dyadic transpose is the case of an array *A*, such that ρ*A* ↔ 5 10 12. The three axes correspond to an account extending over 5 years, for 10 line items, and 12 months. If *A* is displayed, 5 tables will appear, each with 10 rows and 12 columns.

Now suppose that we wish to have a separate table for each line item, then *B*←2 0 1 ⍉ *A* and ρ*B* ↔ 10 12 5; i.e. there are 10 tables (one for each line item), each with 12 rows (the months) and 5 columns (years).

Or we might have chosen to create the array *C*←1 0 2 ⍉ *A*, so that ρ*C* ↔ 10 5 12; ρ*C* ↔ 10 5 12 i.e. there are 10 tables, each with 5 rows (years) and 12 columns (months).

An Example of the Use of APL in Logic

The most amusing and readable book on Logic was written by Lewis Carroll, the author of "Alice in Wonderland" and "Through the Looking Glass". Although its title is "Symbolic Logic", it has rather little to do with the symbolism found in modern texts on that subject, but Carroll does use formal methods of solving syllogisms and sorites. The original edition is rare, but a reprint by Dover Publications Inc. is readily available.

Carroll defines a "sorites" as a set (literally a heap) of three or more propositions related in such a way that two of them together yield a conclusion, which, taken with another of them, yields another conclusion; and so on until all have been taken. If the original set is true then the last conclusion must also be true.

One of Carrolls's examples (#47 on p. 120) is as follows:

(1) Every idea of mine, that cannot be expressed as a Syllogism, is really ridiculous;
(2) None of my ideas about Bath-buns are worth writing down;
(3) No idea of mine, that fails to come true, can be expressed as a Syllogism;
(4) I never have any really ridiculous idea, that I do not at once refer to my solicitor;
(5) My dreams are all about Bath-buns;
(6) I never refer to any idea of mine to my solicitor, unless it is worth writing down.

The "Universe of Discourse" is "My ideas", and the "Dictionary of Terms" is:

1. able to be expressed as a Syllogism
2. about Bath-buns
3. coming true
4. dreams
5. really ridiculous

6. referred to my solicitor
7. worth writing down

Using negative numbers to represent "Not", we can write the 6 bilateral propositions in this way:

```
¯1   5
 2  ¯7
¯3  ¯1
 5   6
 4   2
 6   7
```

Thus the pair ¯1 5 can be read:

All my ideas that are	NOT	able to be expressed as a Syllogism
are all		really ridiculous

and the pair 2 ¯7 can be read:

All my ideas that are		about Bath-buns
are all	NOT	worth writing down

This is an example where origin 1 is preferred to origin 0, because we wish to distinguish between two related cases (true and false) by the use of the sign of the number, and we therefore cannot use 0 for one of our cases.

Now if the proposition ¯1 5 is true, then the proposition ¯5 1 must also be true.

All my ideas that are	NOT	really ridiculous
are all		able to be expressed as a Syllogism

Hence every proposition can be inverted with a change of signs.

Inspection of the numeric table representing the six propositions shows that, with two exceptions, each number (irrespective of its sign) occurs exactly twice. The two exceptions are the terms that occur in the final conclusion.

To implement the analysis, suppose that the propositions given above are assigned to the variable *P*. Then the complete set of propositions is:

```
      M←P,[0]-⌽P
      M
¯1  5
 2 ¯7
¯3 ¯1
 5  6
 4  2
 6  7
¯5  1
 7 ¯2
 1  3
¯6 ¯5
¯2 ¯4
¯7 ¯6
```

The beginning and end of the sequence are found by the function:

```
    BES:(2=+/N∘.=V)/N←NUB V←|,ω
    ⍝  Beginning and end of sorites

    NUB:((ωιω)=ιρω)/ω

          BES M
3 4
```

The start of the sequence is given by:

```
    STS:ω[;0]ι1↑BESω
    ⍝  Start for sorites

          STS |M
2
          M[2;0]
¯3                        Begin with NOT coming true
```

The end of the sequence is given by:

```
    ENS:ω[;1]ι1↓BESω
    ⍝  End of sorites

          ENS |M
10
          M[10;1]
¯4                        End with NOT dreams
```

Hence ¯3 ¯4 or, alternatively, 4 3; i.e. All my dreams come true.

In order to move from the starting proposition to the end, we can proceed as follows:

```
          U←M[;0]=¯3
          U/M[;1]
¯1
          U←M[;0]=¯1
          U/M[;1]
5
          U←M[;0]=5
          U/M[;1]
6
          U←M[;0]=6
          U/M[;1]
7
          U←M[;0]=7
          U/M[;1]
¯2
          U←M[;0]=¯2
          U/M[;1]
¯4                        That this is the end, is shown by:
      M[ENS|M;1]
¯4
      U←M[;0]=¯4If we proceed, the result is empty:
      U/M[;1]
```

Consequently, we can get the sequence by recursion, stopping either when we reach the known end proposition or when the result is empty:

```
  RSR:(α,(ω[;0]=0⊥α)/ω[;1]RSRω:A=0⊥α:α
  ⍝ Recursive sorites

  SRS:ω[STS|ω;0]RSRω:0,0⍴A←ω[ENS|ω;1]:0
  ⍝ Sorites

      SRS M
¯3 ¯1 5 6 7 ¯2 ¯4
```

The sequence needs to be obtained as a function of *P*, and it must be reversed if it starts with a negative:

```
  IXS:Z:0>1↑Z←SRSω,[0]-⌽ω:-⌽7
  ⍝    Indexes of sequence of sorites

      IXS P
4 2 ¯7 ¯6 ¯5 1 3
```

Finally, to translate the result into English we need the following text data, with 'Not' appended to the end of each line.

```
      DATA
able to be expressed as a SyllogismNot
about Bath-buns                    Not
coming true                        Not
dreams                             Not
really ridiculous                  Not
referred to my solicitor           Not
worth writing down                 Not
```

The function Carroll will then reorder the terms, and rotate where a NOT is to be inserted:

```
  CARROLL:0 ¯4↓(¯4×I<0)⌽α[¯1+|I←IXSω;]
  ⍝  α is text. ω is numeric matrix of propositions

      DATA CARROLL P
dreams
about Bath-buns
Not worth writing down
Not referred to my solicitor
Not really ridiculous
able to be expressed as a Syllogism
coming true
```

The conclusion is: All my dreams come true

Data Bases with Multiple-Line Entries

A surprising number of students wished to create a simple data base system. I had provided a workspace with a fairly powerful simulation of IBM's product "Query-by-Example", and had explained the principles involved in its implementation. The students could use this as a model, both to get experience in using such a system and to understand how such a system can be created in APL. However, I wanted them to write their own functions. Although the lessons I learned in writing the system in Direct Definition are relevant to the subject of this paper, a listing of the functions is too long to include here.

One of the problems encountered by students was how to handle the case where some items consist of multiple-lines. We did not have time to implement a system of arrays of arrays, but the following illustrates what might be considered to be the first step in dealing with the problem.

Consider the following data base:

```
      D
0      0     2     2 1915
1      2     4     1 1860
2      6     3     0 1826
0      9     1     2 1912
2     10     1     0 1830
3     11     1     0 1835
2     12     2     0 1837
0     14     2     2 1909
2     16     4     0 1820
2     20     2     1 1818
```

This matrix represents a bibliographic file giving Author, Title, Type of publication, and Date. With the exception of the date, *D*[;4], the numbers in any column are pointers to supporting character arrays. *D*[;0] points to a table of Authors, *AU*; *D*[;1] points to the START of the Title in a table of Titles, *TI*; *D*[;2] gives the number of lines to be taken from *TI*, so that ¯1++/*D*[;1 2] points to the LAST line of each title in *TI*; and *D*[;3] points to a table of Types, *TYPE*. The supporting character arrays (which were conveniently entered by the use of IBM's Extended APL Editor) are the following for this example:

```
      AU
Shaw, G.B.
Tennyson, A.
Scott, W.
Dickens, C.
```

```
      TI
Good King Charles' Golden Days:
A play for 6 characters.
Idylls of the King:
A poem describing the Life and Times
of King Arthur and the Knights of
```

the Table Round.
The Heart of Midlothian:
being an attempt at a Historical
Tale of the Olden Times.
Major Barbara.
The Antiquary.
Nicholas Nickelby.
The Tale of Two Cities:
Paris and London.
The Doctors Dilemma:
the problems of medical practice.
Waverley:
or Tis 60 Years Since;
the story of what followed the
late attempt at Revolution.
Marmion:
being Prelude to the Battle of Flodden.

```
      TYPE
Novel
Poem
Play
```

The problem, of course, is that whereas the Author, the Type, and the Date each take a single line, the Title takes a variable number of lines. Consequently we must expand the selected Authors by inserting blank lines to maintain compatibility with the multi-line titles. The Type and Date must be expanded in the same way. This is complicated by the fact that when we select from *D*, we rearrange the order of the Titles, and thus change the pattern of multi-lines. However, all the information needed for the solution is in `D[;1 2]`. Note that the result is an "Unnormalized Relation" in Relational Data Base terminology.

The functions needed are as follows:

```
RIX:((ω≠0)/⍳⍴ω)[¯1++\¯1⌽∨⌿(+\ω)∘.=1+⍳+/ω] ⍝ Repeated indexes

CTI:((RIX α[;2])∊ω/⍳⍴α[;2])⌿TI ⍝ Compress titles

EXP:¯1⌽(⍳1↑⍴ω)∊¯1++\α
⍝ Expansion vector needed for compatibility with ω as prescribed by α

PRINT:(V⍀AU[ω/α[;0];]),M,(V←(ω/α[;2])EXP M←αCTIω)⍀ω⌿TYPE[α[;3];],⍕α[;,4]
```

- -

In order to understand how these functions work, consider the following illustrations:

```
      RIX 2 4 3          "Repeated indexes"
0 0 1 1 1 1 2 2 2

      W←D[;0]=2          Author with index 2
      W
0 0 1 0 1 0 1 0 1 1
```

```
      D CTI W            Titles of works by Author 2
The Heart of Midlothian:
being an attempt at a Historical
Tale of the Olden Times.
The Antiquary.
The Tale of Two Cities:
Paris and London.
Waverley:
or Tis 60 Years Since;
the story of what followed the
late attempt at Revolution.
Marmion:
being Prelude to the Battle of Flodden.

      W/D[;2]                                 Numbers of lines in the titles
3 1 2 4 2                                     of works by Author 2.

      V←(W/D[;2]) EXP D CTI W              Expansion vector for compatibility
      V                                                       with the titles.
1 0 0 1 1 0 1 0 0 0 1 0

      V⍀AU[W/D[;0];]                     Expand the list of selected Authors.
Scott, W.


Scott, W.
Scott, W.

Scott, W.



Scott, W.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

1. To print the entire data base, we make the selection with a vector that is all 1's. The scalar 1 will extend as needed for compatibility.

```
      D PRINT 1
Shaw, G.B.         Good King Charles' Golden Days:     Play  1915
                   A play for 6 characters.
Tennyson, A.       Idylls of the King:                 Poem 1860
                   A poem describing the Life and Times
                   of King Arthur and the Knights of
                   the Table Round.
Scott, W.          The Heart of Midlothian:            Novel 1826
                   being an attempt at a Historical
                   Tale of the Olden Times.
Shaw, G.B.         Major Barbara.                      Play  1912
Scott, W.          The Antiquary.                      Novel 1830
Dickens, C.        Nicholas Nickelby.                  Novel 1835
Scott, W.          The Tale of Two Cities:             Novel 1837
                   Paris and London.
```

```
Shaw, G.B.          The Doctors Dilemma:                    Play  1909
                    the problems of medical practice.
Scott, W.           Waverley:                               Novel 1820
                    or Tis 60 Years Since;
                    the story of what followed the
                    late attempt at Revolution.
Scott, W.           Marmion:                                Poem  1818
                    being Prelude to the Battle of Flodden.
```

2. To select all works by Scott (Author 2):

```
      D PRINT D[;0]=2
Scott, W.           The Heart of Midlothian:                Novel 1826
                    being an attempt at a Historical
                    Tale of the Olden Times.
Scott, W.           The Antiquary.                          Novel 1830
Scott, W.           The Tale of Two Cities:                 Novel 1837
                    Paris and London.
Scott, W.           Waverley:                               Novel 1820
                    or Tis 60 Years Since;
                    the story of what followed the
                    late attempt at Revolution.
Scott, W.           Marmion:                                Poem  1818
                    being Prelude to the Battle of Flodden.
```

3. To select all works of poetry (Type 1):

```
      D PRINT D[;3]=1
Tennyson, A.        Idylls of the King:                     Poem  1860
                    A poem describing the Life and Times
                    of King Arthur and the Knights of
                    the Table Round.
Scott, W.           Marmion:                                Poem  1818
                    being Prelude to the Battle of Flodden.
```

4. To select Plays and Novels (Types 0 and 2) published after 1830:

```
      D PRINT (D[;3]∊0 2)∧D[;4]>1830
Shaw, G.B.          Good King Charles' Golden Days:         Play  1915
                    A play for 6 characters.
Shaw, G.B.          Major Barbara.                          Play  1912
Dickens, C.         Nicholas Nickelby.                      Novel 1835
Scott, W.           The Tale of Two Cities:                 Novel 1837
                    Paris and London.
Shaw, G.B.          The Doctors Dilemma:                    Play  1909
                    the problems of medical practice.
```

Conclusion

Some people object to APL, claiming either that it is "mathematical" or "inefficient". APL is rich in symbols, and there are those who seem to have real difficulty in using symbols as a tool of thought. This appears to be the reason why one hears objections to APL's mathematical content. Such people prefer to spell out every detail of a process (as one is obliged to do in a language like FORTRAN), or prefer PLUS to +, because they have not yet learned the lesson that Sylvester so eloquently taught a century ago. It takes time and patience to work with students who have this difficulty, to convince them with simple and meaningful examples that they can learn to use symbols to add power to their thinking. This is an old problem. As I pointed out at the APL Users Conference in 1978, Oughtred faced the same difficulty when he introduced the symbol × for multiplication. Students are greatly helped when they manipulate arrays that are meaningful to them; a student of language can transpose a multidimensional array of verb endings (where the axes are conjugation, tense, singular-plural, person, and individual letters), whereas a student of business will gain a similar experience from transposing an array representing financial accounts (where the axes are year, month, and line-item).

The objection on the grounds of supposed "efficiency" is best answered by demonstrating that practical results can be obtained far more quickly in APL than by any alternative. It is important that students have some experience with another language (preferably FORTRAN or COBOL) because these are still so commonly used. But, although some will be compelled by an employer to work in these languages, there will be few who will do so by choice after they have used APL. Moreover, if machine efficiency is really the issue, then it is hard to see why the programs are not written in machine language. I included an introduction to Assembler in my course, so that students would know what this means and would understand a little better what goes on inside the machine. But for a Biologist, an Economist, or other scientist or humanist to be working at any level other than APL seems to me to be where real inefficiency of valuable resources is serious.

Quite apart from its obvious utility as an aid in our solution of practical problems, APL can claim a key position in the curriculum of a Liberal Arts College. Benjamin Franklin, among others, said that man is a tool-making animal. The modern computer is the most powerful and universal tool so far created, and as such its study has a special importance. APL is not only the best way to learn about computers, for example with APL the student can design and operate a simple computer of his own, but it is part of the historical tradition and evolution of mathematics and symbolic thinking. This is what makes APL an exciting subject that should be included wherever the Liberal Arts are taught and respected.

Acknowledgements

I am grateful to K.E. Iverson (I.P. Sharp Associates) and P.B. Yale (Pomona College) for suggestions that have been incorporated in this paper.

Bibliography

Bergquist, J.W. — Algebraic Manipulation. Proc. 6th International APL Users Conference, Anaheim, California. May 14-17, 1974. p.45-49.

Berry, P. et al. — APL and Insight. 1978. APL Press.

Carroll, Lewis — Symbolic Logic. 1958 reprint. Dover Publications, Inc.

Douglas, Paul H. — The Cobb-Douglas production function once again: its history, its testing, and some new empirical values. Journal of Political Economy, 84 (1976) 903-915.

Falkoff, A.D. and D.L. Orth — Development of an APL Standard. APL Quote Quad 9, No.4-Part 2 (June 1979), p.409-453.

Iverson, K.E. — Elementary Analysis. 1976. APL Press.

Iverson, K.E. — APL in Exposition. 1976. APL Press.

Iverson, K.E. — Introducing APL to Teachers. 1976. APL Press.

Iverson, K.E. — Programming Style in APL. Proc. APL Users Meeting, Toronto, 1978, p.200-224. I.P. Sharp Associates.

McIntrye, D.B. — Experience with Direct Definition One-liners in Writing APL Applications. Proc. APL Users Meeting, Toronto, 1978, p.281-297. I.P. Sharp Associates.

McIntyre, D.B. — The Architectural Elegance of Crystals made Clear by APL. Proc. APL Users Meeting, Toronto, 1978, p.233-250. I.P. Sharp Associates.

Poole, Robert W. — Quantitative Ecology. 1974. McGraw-Hill, Inc.

Sykes, Jr., Roy A. — Collected Whiz Bangs: An Anthology of Tutorials on APL Programming Techniques, Vol.1, 1978. STSC, Inc.

PILOT RATES OF PAY - A DATA BASE APPLICATION

Jill A. Kastris
Air Line Pilots Association
Washington, D.C.

Introduction

PROROC stands for Pilot Rates of Compensation, and is the name ALPA has given to our Pilot Rates of Pay data base. The PROROC system was designed for two purposes:

1) To produce a uniform series of tables displaying monthly and hourly pilot rates of pay, and

2) To store the maximum monthly pay rates for a pilot (also called endrates) on a file, so that they can be used in other applications.

A typical PROROC table appears in Figure 1.

This presentation covers the design and structure of the PROROC system, with emphasis on the procedure used to create contract files, input data and obtain the PROROC tables.

How Pilot Rates of Pay are Calculated

Each airline negotiates a labor agreement with its pilot group. These agreements vary in duration, the number of times pilots will receive raises (effective dates) over the life of the agreement and in the method by which pilots are compensated.

Airline pilots are essentially paid an hourly rate based on their status (Captain, First Officer or Second Officer), their length of service with the airline (seniority), the type of aircraft that they fly, and whether they are flying a domestic or international route. The rate of pay also varies depending on the designated speed and weight of the aircraft, and whether the pilot is flying during the day or at night.

Any combination of those, and other parameters can constitute a pilot's monthly rate of pay. The following examples display 3 different formulas for calculating the monthly pay rate (also called an **endrate**) for domestic captains.

Example 1

```
CADEND←CADHRS[I]×(LONG+(DAYRATE×DAYRATIO)+(NITERATE×NITERATIO)+(SPEED×MILERATE)+(WGT×WGTRATE))
```

Example 2

```
CADEND←BASE+CADHRS[I]×((DAYRATE×DAYRATIO)+(NITERATE×NITERATIO)+(SPEED×MILERATE)+(WGT×WGTRATE))
```

Example 3

```
CADEND←CADHRS[I]×((DAYRATE×DAYRATIO)+(NITERATE×NITERATIO))
```

In example 1, the Captain endrate is calculated by multiplying the monthly pay hours by the sum of the hourly rates for his seniority with the airline (LONG), the rate of flying during the day and night, and the designated rates of pay based on the speed and weight of the equipment being flown.

Example 2 is the same as example 1, except that instead of receiving an hourly rate of pay based on seniority, the Captain receives a monthly base pay (BASE), which varies by seniority.

Finally, in example 3, the pilot's seniority and the speed and weight of the aircraft are ignored. The pilot simply gets paid depending on how many hours he flew the plane during the day or at night.

In all, we have uncovered 27 different combinations by which domestic Captain endrates are calculated. These formulas, along with the formulas used to calculate First and Second Officer endrates use a common set of variables, which we call Acronyms. Figure 2 displays the acronym list used in calculating pilot endrates.

Whenever a new contract is signed, a new set of pilot endrates must be calculated. The PROROC system was designed to expedite this task. There are currently 73 contract files on the PROROC system.

Before I continue my discussion of the PROROC data base, I would like to stress that the PROROC system was designed to be **user-oriented**, not computer-oriented. Thus, a minimum amount of APL knowledge is required to use the PROROC system, and all programs are conversational.

Creating a Contract File

The PROROC data base is maintained from 2 functions; *CREATE* and *INPUT*. The *CREATE* function creates a contract file, appends the directory file and creates a package of general information which is placed in the first component of the contract file. The *INPUT* function is used to enter data and calculate and store the monthly pilot endrate.

The *CREATE* function works in the following manner:

First the user is prompted to input the new contract key, which consists of five specifications and looks like this:

```
    NWA,0680,1,0,1
```

a) The airline code (a three-letter acronym. For example, *NWA* is used for Northwest);

b) The month and year the contract was negotiated (0680 = June, 1980);

c) The job class of the contract. This is a code that identifies which pilots are covered by the contract, as can be seen below:

   1 = the contract covers all pilots

   2 = the contract covers Captains and First Officers only

   3 = the contract covers Second Officers only

d) A revision code. This indicates whether the contract was renegotiated before it was amendable (a rare occurrence);

e) The operation code. This determines whether special operations are covered under a separate agreement (e.g. MAC agreement, Charter agreement, etc.). This code is usually 1, which indicates "normal" operation.

The key displayed is for the Northwest contract negotiated in June, 1980, covering all pilots. The key is assigned to the variable *CONTRACT*.

In order to give the contract file a name, the commas are simply removed from the key via compression. The compressed contract name is assigned to the variable *CNAME*.

Next, the Program checks the file against the directory file to ensure that it is a unique name. This is accomplished not by checking the file name, but by checking the file number associated with the name. This number also serves as a unique file tie number, which activates the contract file.

All of the three letter airline codes have been made into variables in the PROROC workspace and assigned an integer value from 1 to 99. *NWA*, for example, is a variable with the value 20.

Thus, by using a combination of the decode (⊥) and execute (⍎) operators the contract name is converted to a numeric equivalent called *CNUM*, as shown below:

```
CONTRACT ← NWA,0680,1,0,1

CNUM ← 100 1000 10 10 10 ⊥ ⍎ CONTRACT
```

(*NWA* has been assigned the number 20)

In our example, *CNUM* would be assigned the number 200680101.

This number is checked against the directory file. The first component of the file is a vector of all contract file numbers. If *CNUM* is already in the vector, an error message is printed saying that a file with that name already exists. The *CREATE* program is then exited.

If *CNUM* is not part of the directory, the contract file is created, and *CNUM* is appended to the directory. *CNUM* serves as a unique tie number for the contract file and is used whenever contract files are accessed.

Creating the Package of General Information

Once the contract file has been created, the user is prompted to input some general information about the contract. This information includes the following items:

1) The highest seniority year for which pilots will be paid (usually about 12 years);

2) The amendable date of the contract;

3) The component directory - a matrix of every equipment type/effective date combination for which pilots receive a pay raise;

4) A formula for calculating pilot endrates;

5) The list of variables, called "acronyms", used in calculating pilot endrates.(Actually, this is a vector of acronym numbers that correspond to the various acronyms. There are 91 in all.)

The user is prompted to enter the maximum number of seniority years followed by the amendable date of the contract. After these 2 prompts have been answered, the user must then identify every equipment type/effective date combination for which there are pilot endrates. This is the component directory.

Creating the component directory is a simple task:

PROMPT

HOW MANY EQUIPMENT TYPES DOES THIS CONTRACT HAVE:

USER ENTERS

2

PROMPT

ENTER EQUIPMENT CODE 1

USER ENTERS

BE,00 (for B-747, all models)

PROMPT

```
ENTER ALL EFFECTIVE DATES FOR THIS EQUIPMENT TYPE.
SEPARATE BY SPACES. TYPE 0 IF EFFECTIVE DATES ARE
THE SAME AS THOSE PREVIOUSLY INPUT:
```

USER ENTERS

```
070180 070181 070182
```

PROMPT

```
ENTER EQUIPMENT CODE 2:
```

USER ENTERS

`BC,00` (for B-727, all models)

PROMPT

```
ENTER ALL EFFECTIVE DATES FOR THIS EQUIPMENT TYPE.
SEPARATE BY SPACES.  TYPE 0 IF EFFECTIVE DATES ARE
THE SAME AS THOSE PREVIOUSLY INPUT:
```

USER ENTERS

```
0
```

PROMPT

```
THE FOLLOWING IS THE LIST OF EQUIPMENT CODE/EFFECTIVE DATE
COMBINATIONS YOU HAVE INPUT.  YOU WILL
BE RESPONSIBLE FOR ENTERING
ENDRATE DATA FOR ALL OF THESE COMBINATIONS:

B-747                    070180
B-747                    070181
B-747                    070182
B-727                    070180
B-727                    070181
B-727                    070182

ARE THESE OK (ENTER YES OR NO):
```

USER ENTERS

YES
(If the user had entered *NO*, the entire prompt sequence would have been repeated.)

The user is first asked how many different equipment types there are at the airline. Then he inputs the first equipment type, followed by all the effective dates for pay raises. This is repeated until all equipment type/effective date combinations have been input. (Since pay raise dates are usually the same for all pieces of equipment, the user can repeat the previously typed-in effective dates by typing 0.) Once the matrix is completed, the component directory (called *COMPDIR*) is created, and an empty vector is appended to the contract file for each row in *COMPDIR*. From this point on, the computer will only have to use file functions ⎕*READ* and ⎕*REPLACE* when working with the contract file.

Next, the user must enter the *FORMULA* function for calculating pilot endrates and the component directory. I would like to state again that the PROROC system was designed to be user-oriented, not computer-oriented. Thus, a minimum amount of APL knowledge is required to use PROROC. (In fact, our PROROC user's manual was written by a part-time employee who had no knowledge of APL whatsoever before using the PROROC system.)

Essentially, all that a user needs to know to create *FORMULA* functions are the arithmetic symbols +, -, × and ÷, and the APL definition symbol (←). There are strict rules for creating the formula function. For example, endrate data is always defined as a set of variables ending with the letters *END* (*CADEND* stores the data for Captain, domestic endrates.) The user needs to know how the various acronyms are related.

Rather than force the user to leave the *CREATE* function to create the *FORMULA* function, the computer prompts the user to enter each line of the formula. These lines are appended to a character matrix called *MAT*, whose top row contains the word *FORMULA*. Then, by using the *⎕FD* function the character matrix is converted into an APL function. 3 *⎕FD MAT* will convert *MAT* into a function. The top line of *MAT* becomes the function name.

We found that by taking this approach, the user could even correct lines of the *FORMULA* function that had errors without leaving the *CREATE* function. (Of course, if an error is uncovered, the user must retype the entire line. But a line of the *FORMULA* function is generally short, so it was judged to be a small price to pay.)

Finally, the user is prompted to identify all the acronyms for which he will have to enter data. Rather than having to type in their names, each acronym has been assigned a code number, as shown in Figure 2. Thus, the user need only enter the numbers of the acronym he wishes to use:

PROMPT

ENTER ALL ACRONYM NUMBERS FOR WHICH YOU DESIRE TO INPUT DATA.
SEPARATE BY SPACES:

USER ENTERS

43 9 10 53 74 48, etc.

The user can also take advantage of the *COPY* feature. Although pay formulas vary by airline, they rarely change when an airline negotiates a new pilot contract. Why then should a user have to waste time retyping the same formula function?

The *CREATE* function, therefore, gives the user the option of copying the formula function from another contract file. If the user chooses this option, all that has to be typed in is the additional contract key. The *FORMULA* function **and** the acronym list from the old contract file's first component are unpacked and displayed for the user to edit. (It is assumed that if the formula didn't change, the acronyms required remained the same.)

Let's re-examine the total set of general information the user must input.

Contents of the Contract File General Information Package

Item Number	Description	Variable or Function Name	Type of APL Result
1	The highest seniority year for which pilots will be paid	*CYRS*	Scalar-numeric
2	the amendable date of the contract.	*AMDATE*	Character vector
3	The component directory.	*COMPDIR*	Numeric matrix
4	A formula for calculating pilot endrates.	*FORMULA*	Function
5	A list of acronym numbers corresponding to the acronyms used in calculating pilot endrates.	*ACRNOS*	Numeric vector

In examining this list, one can see how unalike these items are. In APL terms, the general information consists of a scalar integer, a character vector, a two-column integer matrix (each row is a unique equipment type/effective date combination), an APL function and a numeric vector.

This is an ideal application for packages. A package is a method by which various items can be combined into a single unit for later use. Although they take up room in a workspace, packages have no apparent size or shape. In fact, the only actions that packages will allow are, essentially, adding items to the package, finding out what is in the package and, of course, activating the items in the package (unpacking). Further, packed items retain their variable **names** as well as their values.

By packing the previously mentioned items into the first component of the contract file, we were able to save space in the file, store the items conveniently and keep all the general information in one place.

The *COPY* feature previously mentioned is evidence of the advantages of packing related items together. Rather than having to search a file for 2 components, all that needs to be done to activate a previously input *FORMULA* function and acronym list is the following statement:

```
'ACRONOS FORMULA' ⎕PDEF ⎕READ OLDCNUM,1
```

OLDCNUM = the contract file being copied

ACRNOS = the vector of acronym numbers

FORMULA = the old formula function

Once all of the general information has been input, it is packed into the first component of the contract file and the *CREATE* function is completed. The entire process usually takes no more than 30 minutes to complete.

Inputting Data

Once the contract file and general information have been established, the user can input the data needed to create the PROROC tables. The following steps are required to obtain the tables:

1) Identify the contract file;

2) Identify the specific equipment type/effective date combination desired;

3) Check and correct the *FORMULA* function and acronym listing (optional);

4) Input data for all acronyms or for a specific set of acronyms;

5) Correct any errors in data input;

6) Add "free form" footnotes for printing (optional);

7) Check the maximum rates of pay table (printed by the computer) against hand calculated results;

8) Identify a new equipment type/effective date combination;

9) Modify the previously input data so that they are correct for the new combination.

The program that is accessed is called *INPUT*. Identifying the contract file is done in the same way as in the *CREATE* function. The user inputs the airline code, negotiation ID, job class, revision code and operation code separated by commas. Upon input, the contract file is tied and the first component, which contains the general information, is unpacked and made part of the workspace.

The user then inputs the equipment code/effective date combination desired. This is checked against the component directory. If the combination does not exist an error message is printed and the user gets another chance. If the combination was acceptable, the *INPUT* program will proceed to the next prompt.

Since the user has the option of retaining previously input data (explained later), the next prompt asks if this option was exercised. If it was, then only some of the data needs to be modified, and the program branches to step 4. Otherwise, the user gets the chance to examine and modify the *FORMULA* function and the acronym list one last time. The user is now ready to input the data for the first set of pilot endrates.

Figure 2 displays the list of all acronyms acceptable to the PROROC system and their corresponding numbers. Actually, all of the acronyms are simply rows of a character matrix. There are **no** data variables in the PROROC workspace. The *INPUT* program creates them. Data variables are created by using the execute operator (⍎) in combination with the rho (⍴) and take (↑) operators and the maximum seniority years (*CYRS*)variable from the contract file. The line that creates the acronym data looks like the APL statement below.

STEP 1: The user inputs the data for the variable as prompted.

STEP 2: The computer converts the input data into a variable using the following statement:

```
⍎ (,ACRONYMS[ACRNOS[I]:]),'← CYRS ↑ CYRS ρ ¯1 ↑ IN'
```

where *IN* is the data input by the user.

This ensures that a vector of data equal to the number of contracted seniority years will always result. It also allows the user the convenience of not having to repeat data entries if they are less than the number of seniority years. For example, if the total number of seniority years is 12, but First Officer pay percentages only go through 8 years of service, the user only has to input 8 numbers. The last number input will be carried through the additional 4 years.

Example Of Entering Data

PROMPT

```
INPUT FOPCT:
```

USER ENTERS

```
0 .50 .55 .56 .57 .58 .59 .60
```

If there are 12 seniority years in the contract, the computer will execute the following statement:

```
FOPCT←0 .50 .55 .56 .57 .58 .59 .60 .60 .60 .60 .60
```

Some of the acronyms require that only 5 numbers be input by the user. These are used to calculate the 5 pilot endrates that appear on the PROROC tables. Most of these acronyms, in fact, are for the hours of service for which monthly endrates will be calculated. The *INPUT* program identifies these acronyms and ensures that only 5 numbers are input. Otherwise, the variables are created in the same manner just described.

By using the execute operator to create the acronym variables, the PROROC workspace is kept clean, insuring that no extraneous data will be used in calculating the endrates (and also guarding against workspace full errors).

After all the data has been input, the user is asked if the inputs were correct. If they were not, the user simply inputs the acronym numbers of the incorrect data, and re-enters the correct data when prompted.

A "datadump" is then printed. This displays all of the data just input, the *FORMULA* function, the contract file name and the equipment code/effective date combination on one page. The user has the option of suppressing printing at a later date (this will be discussed in more detail later). We have found the "datadumps" to be quite useful, since virtually all of the information needed to calculate an endrate is now on a single page. A copy of a typical "datadump" can be seen in Figure 3.

After the datadump is printed, the user gets another chance to correct the data as previously described. If all is OK, the *INPUT* program asks if any footnotes are to be added.

Footnotes are used to describe something unusual about a particular endrate. A maximum of 3 lines can be used for footnotes. The user is limited to 100 spaces per line.

Footnotes are stored as variables ending in the letters *FNOTE*. *CADFNOTE* would be the footnote for domestic captain endrates. These footnotes are printed at the bottom of all PROROC tables, and remain the same until changed by the user.

At this point, the PROROC tables are ready to be printed. The user is asked if he wishes to have the tables printed at the terminal or stored for later printing. Each table takes about 2 minutes to print. Since up to 6 tables can be printed for a given equipment type/effective date combination, the choice is usually made to store the printouts on a file.

Finally, a table of maximum pilot endrates is printed. This is the final check for the user. If these numbers match the hand-calculated values the user has computed, the PROROC tables are printed. If not, the *INPUT* program ends and the user is instructed to rename the workspace, save his data and sign off until the problem can be solved. (We have found the maximum endrate tables quite useful. In fact, we tape them to the appropriate "datadump" pages to add to their value.)

If the user accepts the endrates, the PROROC tables are printed (or stored on a file), the pilot endrates for the monthly maximum hours flown are stored in the appropriate component on the contract file, and the user is asked if he wishes to continue inputting data for the contract. If he does, the new equipment code/effective date combination is input, and the user is asked if the current data in the workspace is acceptable.

Generally only a few of the acronyms change with any given combination. Most of the data remains the same. Thus, the user does not have to re-input all of the data. He only identifies the acronym numbers of the data that changes and then inputs the corrected data when prompted. This cuts the time it takes to input data considerably. For example, the current Northwest contract has 16 equipment code/effective data combinations. All data was input for these combinations in a little more than one hour.

If the user decides to take a break, he can state that he no longer wishes to input data. The computer will then instruct the user to rename the workspace, save it and sign off. At a later time, the user need simply reload the saved workspace and type *INPUT*. After identifying the contract file, he can instruct the program that he wishes to modify only some of the data, at which time, the *INPUT* program will branch to the point just prior to printing the "datadump" page, and ask which acronyms need to be modified.

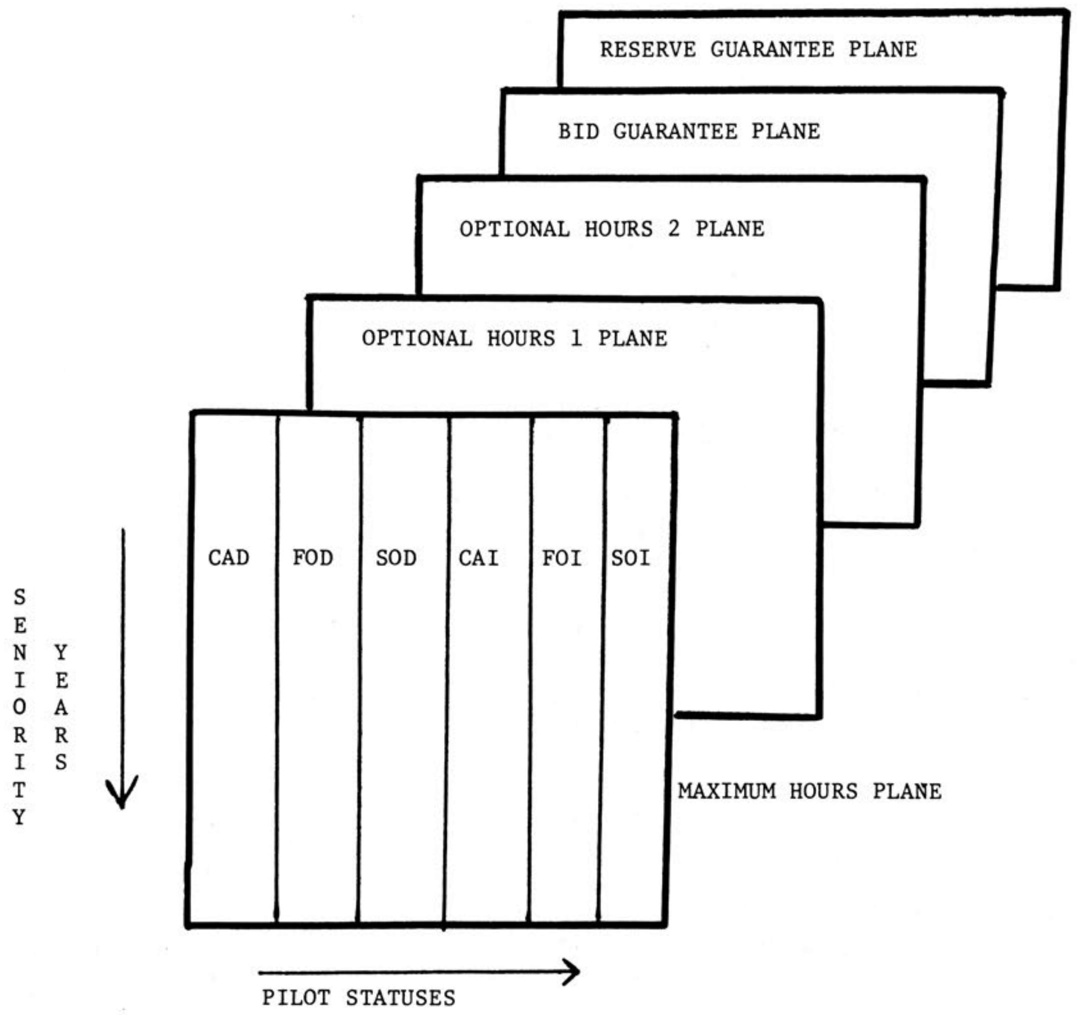
Calculation of Pilot Endrates

Once the user is satisfied that the data is correct, the *INPUT* program calculates the pilot endrates. As I said earlier, and we have seen by examining some *FORMULA* functions, pilots are essentially paid an hourly rate that varies depending on a pilot's seniority, his status, the time of day he is flying and the weight and speed of the aircraft.

Although this data is represented as an hourly rate of pay, pilots are actually paid monthly. Let's re-examine a finished PROROC table (Figure 1). Note that 5 sets of endrates appear. The leftmost column is the maximum monthly endrate a pilot can earn. In this case, the endrate is for 75 hours of flying during the month. (Generally, the maximum monthly hours will be between 75 and 85 hours.) The 4th and 5th columns are under the heading "Guarantees". Pilots who fly a preset schedule are called Lineholder pilots. Pilots who do not fly predetermined routes, but can be called upon at any time to fly the line are called Reserve pilots.

Both types of pilots are guaranteed a minimum rate of pay. In the table in Figure 1, lineholder pilots are guaranteed 63 hours pay per month and reserve pilots are guaranteed 70 hours.

In all, five sets of pilot endrates are calculated, one each for the maximum pay hours, the bid and reserve guaranteed hours, and 2 optional hours. The endrate data is stored in a 3-dimensional array. Each plane of the array contains the endrates for a given set of hours; each row contains the endrates for a given seniority year; and each column is a pilot status (captain domestic, first officer domestic, etc.)



CAD = Captain Domestic
FOD = First Officer Domestic
SOD = Second Officer Domestic
CAI = Captain International
FOI = First Officer International
SOI = Second Officer International

Pilot Endrate Array

The number of rows (seniority years) is determined by the *CYRS* variable from component 1 of the contract file.

The number of columns depends on both the job class, which is part of the contract name, and the type of operation. Type of operation is determined by the *ENTITY* acronym, which is a mandatory input for all pilot contracts. *ENTITY* is a number from 1 to 3:

1 = domestic operations only

2 = international operations only

3 = both domestic and international operations

The number of columns in the endrate array is based on job class and entity.

Job/Class	**Entity**		
	1	**2**	**3**
1	3	3	6
2	2	2	4
3	1	1	2

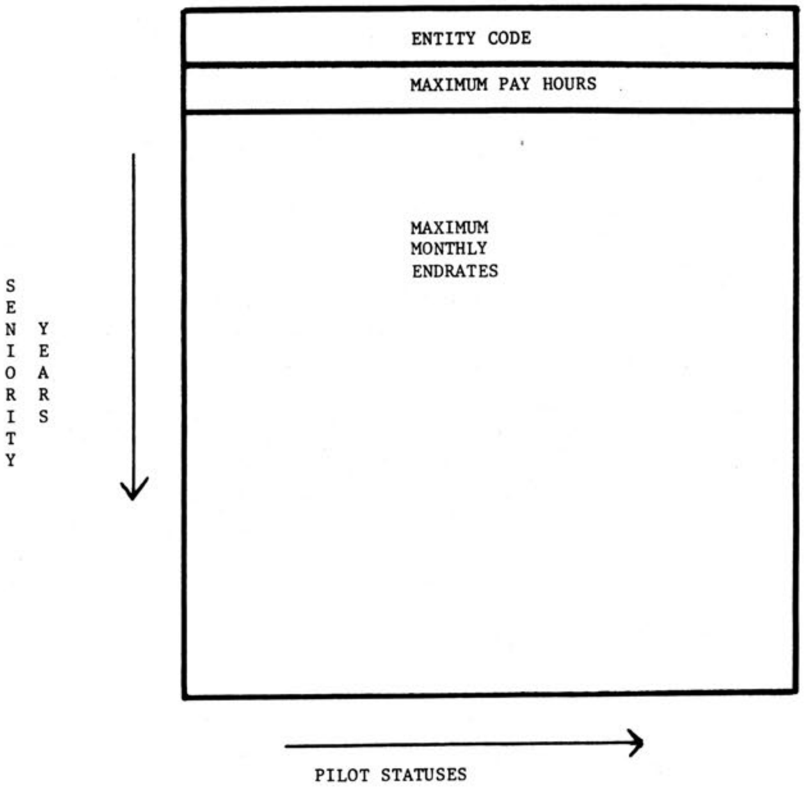
The order of the columns is always Captains, First Officers, Second Officers. All domestic rates are always printed before international rates.

A subroutine called *CALC* identifies the size of the endrate array and calculates the endrates using the *FORMULA* function. The *CALC* function also creates a *FLIGHT FLIGHT PAY* array, similar in size to the *ENDRATE* array.

To print the PROROC tables, a column of the endrate array is transposed into a 5-column matrix. The first column (maximum monthly endrates) is divided by the maximum hours to obtain an hourly rate of pay and appended to the end of the matrix. The corresponding column of the first plane of the *FLIGHT PAY* array is then added to the end of the matrix.

Once the PROROC matrix has been formed, it is formatted with appropriate titles and footnoted, so that the end result looks like the table shown in Figure 1.

After all the PROROC tables have been printed, the first plane of the endrate array, which contains the maximum monthly endrates is transformed into a matrix. Then the *ENTITY* code and the maximum pay hours are added to the top of the matrix as illustrated below.



Endrate Matrix Stored on Contract File

This matrix is then stored in the appropriate component on the contract file. The component directory is used to send the data to the correct place on the file.

Storing and Printing the PROROC and DATADUMP Tables

As I said earlier, the user can opt to suppress printing to PROROC and DATADUMP tables in order to expedite input of data. Since they take about 2 minutes a page to print, this is a necessary step. Each equipment code/effective date combination can result in six PROROC tables being printed. Thus 12 minutes of time can be wasted while these tables are printing. More important, by storing the tables, they do not have to be recalculated if a line hit occurs or if the system goes down.

The PROROC file stores PROROC tables. The *RAWDATA* file stores the "datadump" tables. To facilitate printing, a *SWITCH* function was created to move the "datadumps" to the back of the PROROC file.

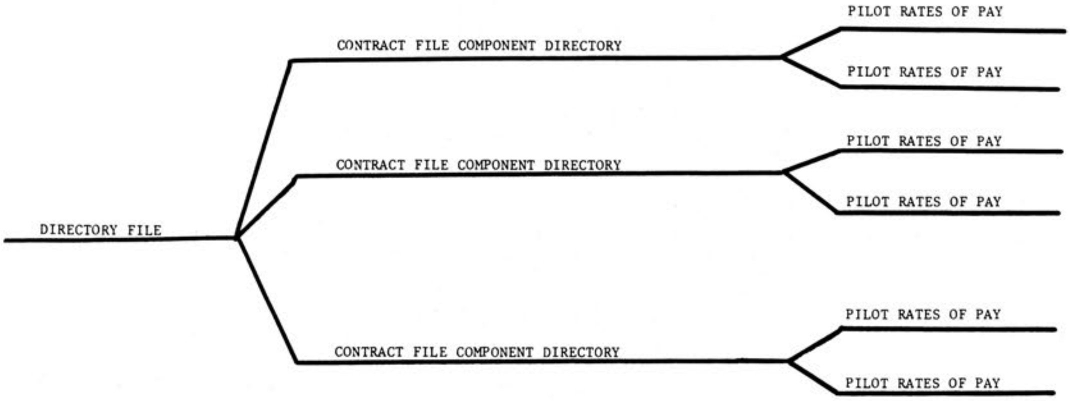
Back in 1978, when we first created the PROROC system, we had to input the data for all current pilot contracts. This led to a mammoth PROROC file that was literally unprintable at the terminal. To give you a small example of the size of the file, the United contract had 30 equipment type/effective date combinations, which resulted in 180 PROROC tables. At two minutes a page, this contract **alone** would take six hours to print out at the terminal (not counting another hour for the "datadumps"). Indeed, at one point, we had over 1000 PROROC tables on the file. Needless to say, we took a crash course in Sharp's highspeed print facilities.

Our PROROC and datadump files were already formatted and paged exactly as we wanted them to appear. Thus we did not want the "canned" paging and date stamping routines which are standard with Sharp's HSPRINT. In order to circumvent these we simply had to create two components at the beginning of the print file. The first component contains the scaler integer two (2). This suppressed all of the extraneous printing. The second component was a vector of 66 1's. This told the printer to ignore the canned paging and print 66 lines on each page, regardless of the file component, thus acknowledging the fact that our PROROC tables contained "built-in" paging.

In order to execute the highspeed print option, the user must first execute the *SWITCH* function, which combines the PROROC and RAWDATA files. The combined file is then renamed PRINTOUT, and the old PROROC and RAWDATA files are emptied. This allows another user to begin inputting new data. The PRINTOUT file is automatically erased upon completion of the highspeed print, saving storage costs. Since this creates potential problems due to the mail, we generally wait 24 hours before running an HSPRINT, to ensure the fact that the files can always be reactivated.

General Design

Following is the structure of the PROROC data base:



The design is a standard "tree" structure. The data base is accessed as follows:

STEP 1: The user inputs the contract key (identifier) desired.

STEP 2: The computer checks this key against the directory file to ensure that a contract file exists. If the contract key does not appear on the file, an error message is printed and the user is asked to try again. If the contract key is on the directory file, the contract file is activated (tied).

STEP 3: The user inputs the desired equipment code/effective date combination.

STEP 4: The contract file consists of one component for each equipment type/effective date combination, plus a component containing a package of general information. Component 1 of the file contains the package of general information, which includes a component directory. This is a matrix of all equipment type/effective date combinations for which pilot endrates can be calculated. Each row of the component directory corresponds to a component of the contract file. (The first row of the component directory corresponds to the second component, and so on).

The computer checks the equipment code/effective date combination input by the user against the component directory. If the combination is not in the component directory an error message is printed. If the combination exists, the appropriate file component is activated and brought into the workspace.

(Note: this component directory is also used to determine where endrates are stored when they are originally calculated.)

The rates of pay file components are a series of matrices which were previously described.

Conclusion

In concluding, I would like to point out that the PROROC system takes advantage of two of APL's strongest attributes - the speed in which a system can be developed and the ease with which programs can be modified.

Two people developed the PROROC system from scratch. Within 6 weeks from when they started, we had completed PROROC tables for all of the contracts that were in effect at the time. Further, debugging a program or adding a feature to the printouts never took more than one hour.

Finally, by making the system user-oriented, we kept required knowledge of APL at a minimum. This made training new users fast, and eliminated virtually all potential errors.

NORTHWEST
PILOT RATES OF PAY

DATE CONTRACT NEGOTIATED: 08-78
DATE CONTRACT AMENDABLE: 06-30-80

UNION: ALPA

STATUS: FIRST OFFICER
OPERATION: INTERNATIONAL

EQUIPMENT TYPE: B-727 ALL MODELS
EFFECTIVE FROM: 07-01-78
TO: 06-30-79

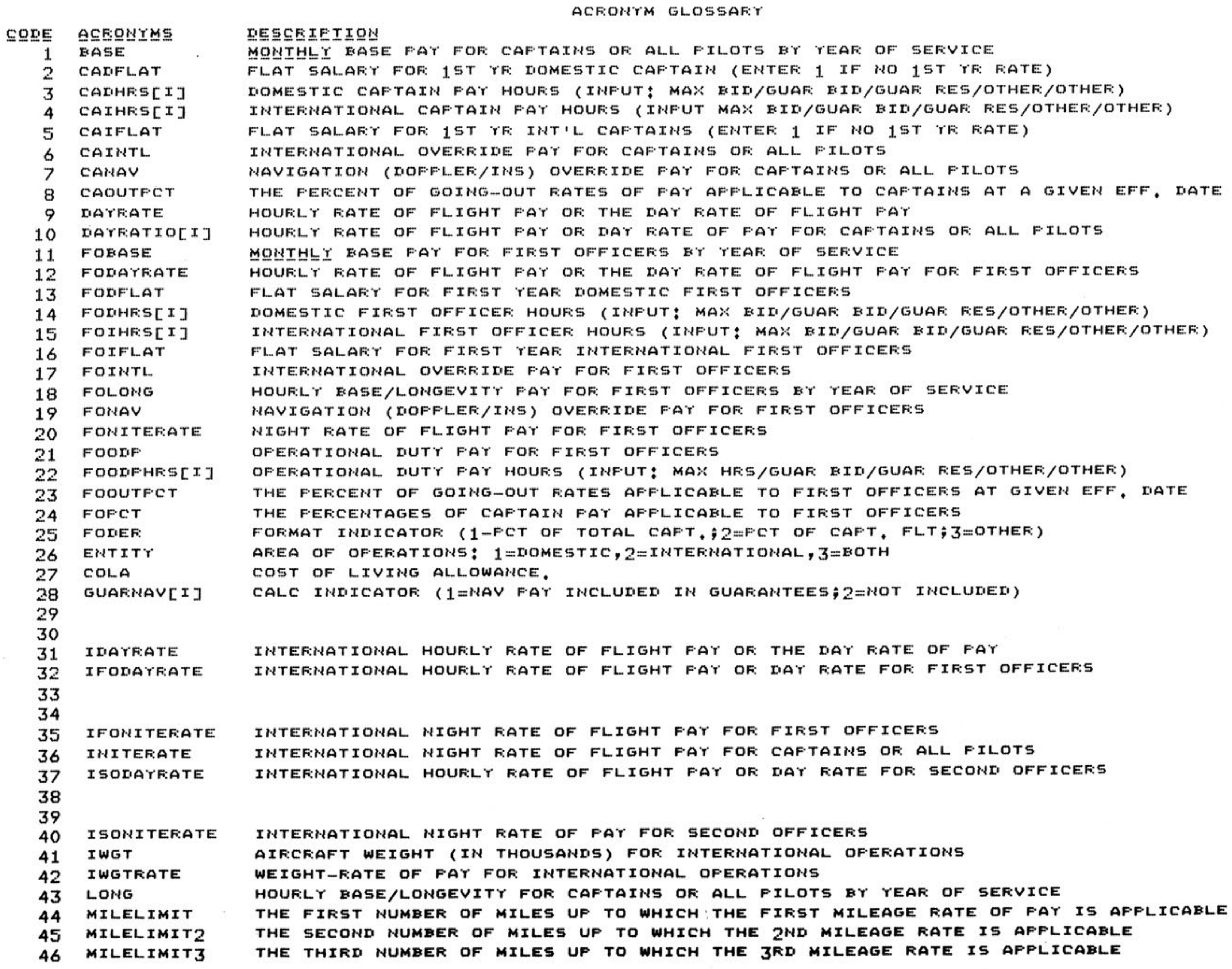
	MONTHLY GROSS PAY YIELDS			GUARANTEES		PAY RATES PER HOUR		
	MAXIMUM			LINEHOLDER	RESERVE	MAX MONTHLY		
	HOURS 75	HOURS 80	HOURS 86	HOURS 63	HOURS 70	GROSS YIELD	FLIGHT PAY	
YEAR	RATIO 50/50	RATIO 50/50	RATIO 50/50	RATIO 50/50	RATIO 50/50	PER HOUR	PER HOUR	YEAR
1	1,000	1,000	1,000	1,000	1,000	13.33		1
2	2,858	3,049	3,238	2,275	2,528	38.11	36.19	2
3	3,454	3,684	3,914	2,775	3,084	46.05	43.03	3
4	3,536	3,772	4,008	2,845	3,161	47.15	43.71	4
5	3,622	3,863	4,105	2,916	3,240	48.29	44.40	5
6	3,707	3,955	4,202	2,988	3,320	49.43	45.08	6
7	3,793	4,046	4,268	3,060	3,400	50.57	45.76	7
8	3,873	4,131	4,389	3,127	3,475	51.64	46.45	8
9	3,986	4,245	4,511	3,217	3,575	53.07	47.47	9
10	4,055	4,326	4,596	3,280	3,645	54.07	48.16	10
11	4,102	4,375	4,649	3,320	3,689	54.69	48.50	11
12	4,145	4,421	4,697	3,356	3,729	55.26	48.50	12

NOTE:

FIRST OFFICERS RECEIVE $2.00 PER HOUR NAVIGATIONAL PAY.
FIRST OFFICERS GET A PERCENTAGE OF TOTAL CAPTAIN PAY.
FLIGHT PAY PER HOUR INCLUDES ONLY HOURLY, WEIGHT, SPEED, INTERNATIONAL AND DOPPLER WHEN APPLICABLE.

PREPARED: 8/25/78
RESEARCH DEPARTMENT
AIR LINE PILOTS ASSOC.

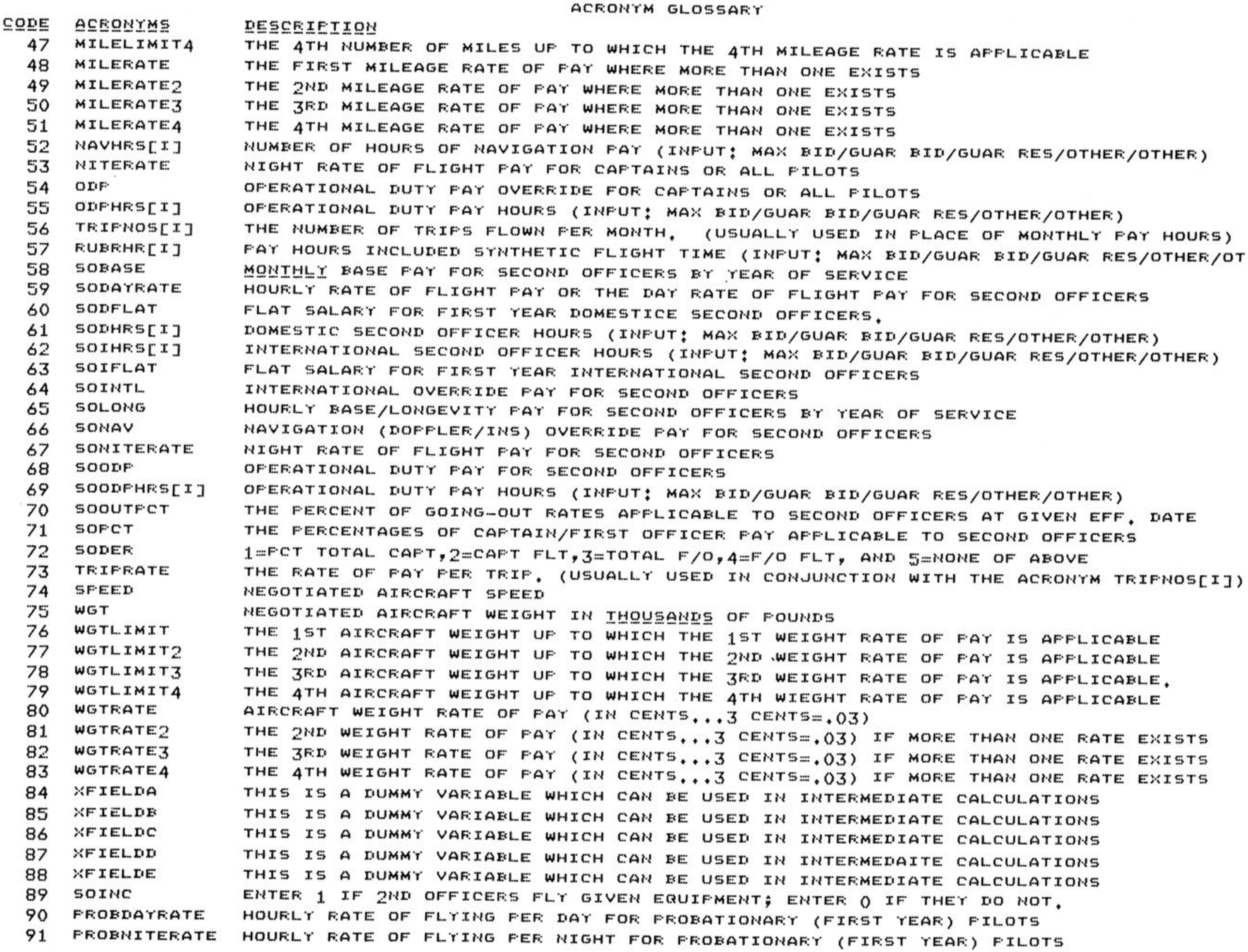
Figure 1



ACRONYM GLOSSARY

CODE	ACRONYMS	DESCRIPTION
1	BASE	MONTHLY BASE PAY FOR CAPTAINS OR ALL PILOTS BY YEAR OF SERVICE
2	CADFLAT	FLAT SALARY FOR 1ST YR DOMESTIC CAPTAIN (ENTER 1 IF NO 1ST YR RATE)
3	CADHRS[I]	DOMESTIC CAPTAIN PAY HOURS (INPUT; MAX BID/GUAR BID/GUAR RES/OTHER/OTHER)
4	CAIHRS[I]	INTERNATIONAL CAPTAIN PAY HOURS (INPUT MAX BID/GUAR BID/GUAR RES/OTHER/OTHER)
5	CAIFLAT	FLAT SALARY FOR 1ST YR INT'L CAPTAINS (ENTER 1 IF NO 1ST YR RATE)
6	CAINTL	INTERNATIONAL OVERRIDE PAY FOR CAPTAINS OR ALL PILOTS
7	CANAV	NAVIGATION (DOPPLER/INS) OVERRIDE PAY FOR CAPTAINS OR ALL PILOTS
8	CAOUTPCT	THE PERCENT OF GOING-OUT RATES OF PAY APPLICABLE TO CAPTAINS AT A GIVEN EFF. DATE
9	DAYRATE	HOURLY RATE OF FLIGHT PAY OR THE DAY RATE OF FLIGHT PAY
10	DAYRATIO[I]	HOURLY RATE OF FLIGHT PAY OR DAY RATE OF PAY FOR CAPTAINS OR ALL PILOTS
11	FOBASE	MONTHLY BASE PAY FOR FIRST OFFICERS BY YEAR OF SERVICE
12	FODAYRATE	HOURLY RATE OF FLIGHT PAY OR THE DAY RATE OF FLIGHT PAY FOR FIRST OFFICERS
13	FODFLAT	FLAT SALARY FOR FIRST YEAR DOMESTIC FIRST OFFICERS
14	FODHRS[I]	DOMESTIC FIRST OFFICER HOURS (INPUT; MAX BID/GUAR BID/GUAR RES/OTHER/OTHER)
15	FOIHRS[I]	INTERNATIONAL FIRST OFFICER HOURS (INPUT; MAX BID/GUAR BID/GUAR RES/OTHER/OTHER)
16	FOIFLAT	FLAT SALARY FOR FIRST YEAR INTERNATIONAL FIRST OFFICERS
17	FOINTL	INTERNATIONAL OVERRIDE PAY FOR FIRST OFFICERS
18	FOLONG	HOURLY BASE/LONGEVITY PAY FOR FIRST OFFICERS BY YEAR OF SERVICE
19	FONAV	NAVIGATION (DOPPLER/INS) OVERRIDE PAY FOR FIRST OFFICERS
20	FONITERATE	NIGHT RATE OF FLIGHT PAY FOR FIRST OFFICERS
21	FOODP	OPERATIONAL DUTY PAY FOR FIRST OFFICERS
22	FOODPHRS[I]	OPERATIONAL DUTY PAY HOURS (INPUT; MAX HRS/GUAR BID/GUAR RES/OTHER/OTHER)
23	FOOUTPCT	THE PERCENT OF GOING-OUT RATES APPLICABLE TO FIRST OFFICERS AT GIVEN EFF. DATE
24	FOPCT	THE PERCENTAGES OF CAPTAIN PAY APPLICABLE TO FIRST OFFICERS
25	FODER	FORMAT INDICATOR (1-PCT OF TOTAL CAPT.;2=PCT OF CAPT. FLT;3=OTHER)
26	ENTITY	AREA OF OPERATIONS; 1=DOMESTIC,2=INTERNATIONAL,3=BOTH
27	COLA	COST OF LIVING ALLOWANCE.
28	GUARNAV[I]	CALC INDICATOR (1=NAV PAY INCLUDED IN GUARANTEES;2=NOT INCLUDED)
29		
30		
31	IDAYRATE	INTERNATIONAL HOURLY RATE OF FLIGHT PAY OR THE DAY RATE OF PAY
32	IFODAYRATE	INTERNATIONAL HOURLY RATE OF FLIGHT PAY OR DAY RATE FOR FIRST OFFICERS
33		
34		
35	IFONITERATE	INTERNATIONAL NIGHT RATE OF FLIGHT PAY FOR FIRST OFFICERS
36	INITERATE	INTERNATIONAL NIGHT RATE OF FLIGHT PAY FOR CAPTAINS OR ALL PILOTS
37	ISODAYRATE	INTERNATIONAL HOURLY RATE OF FLIGHT PAY OR DAY RATE FOR SECOND OFFICERS
38		
39		
40	ISONITERATE	INTERNATIONAL NIGHT RATE OF PAY FOR SECOND OFFICERS
41	IWGT	AIRCRAFT WEIGHT (IN THOUSANDS) FOR INTERNATIONAL OPERATIONS
42	IWGTRATE	WEIGHT-RATE OF PAY FOR INTERNATIONAL OPERATIONS
43	LONG	HOURLY BASE/LONGEVITY FOR CAPTAINS OR ALL PILOTS BY YEAR OF SERVICE
44	MILELIMIT	THE FIRST NUMBER OF MILES UP TO WHICH THE FIRST MILEAGE RATE OF PAY IS APPLICABLE
45	MILELIMIT2	THE SECOND NUMBER OF MILES UP TO WHICH THE 2ND MILEAGE RATE IS APPLICABLE
46	MILELIMIT3	THE THIRD NUMBER OF MILES UP TO WHICH THE 3RD MILEAGE RATE IS APPLICABLE

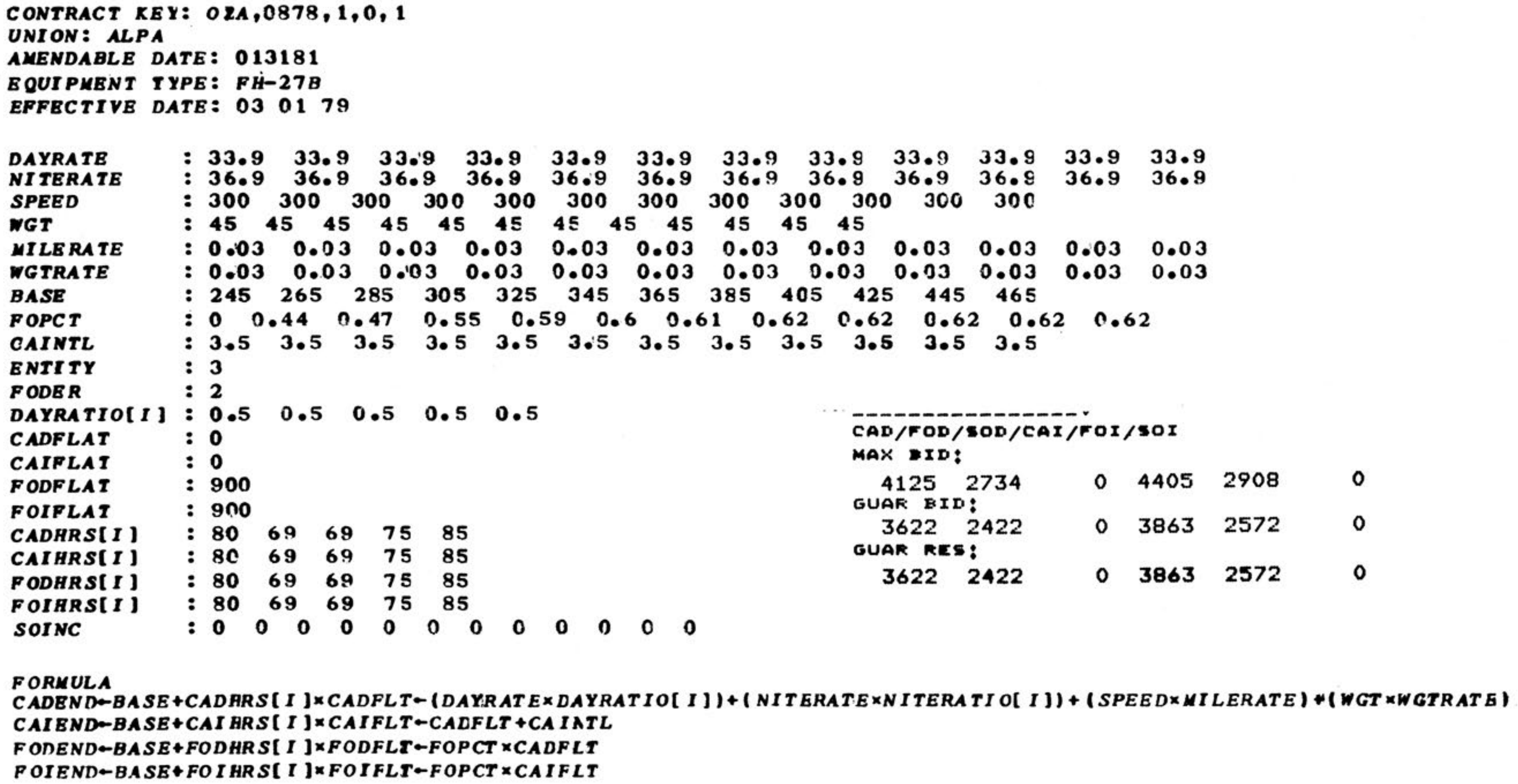
Figure 2 (Part 1)



ACRONYM GLOSSARY

CODE	ACRONYMS	DESCRIPTION
47	MILELIMIT4	THE 4TH NUMBER OF MILES UP TO WHICH THE 4TH MILEAGE RATE IS APPLICABLE
48	MILERATE	THE FIRST MILEAGE RATE OF PAY WHERE MORE THAN ONE EXISTS
49	MILERATE2	THE 2ND MILEAGE RATE OF PAY WHERE MORE THAN ONE EXISTS
50	MILERATE3	THE 3RD MILEAGE RATE OF PAY WHERE MORE THAN ONE EXISTS
51	MILERATE4	THE 4TH MILEAGE RATE OF PAY WHERE MORE THAN ONE EXISTS
52	NAVHRS[I]	NUMBER OF HOURS OF NAVIGATION PAY (INPUT; MAX BID/GUAR BID/GUAR RES/OTHER/OTHER)
53	NITERATE	NIGHT RATE OF FLIGHT PAY FOR CAPTAINS OR ALL PILOTS
54	ODP	OPERATIONAL DUTY PAY OVERRIDE FOR CAPTAINS OR ALL PILOTS
55	ODPHRS[I]	OPERATIONAL DUTY PAY HOURS (INPUT; MAX BID/GUAR BID/GUAR RES/OTHER/OTHER)
56	TRIPNOS[I]	THE NUMBER OF TRIPS FLOWN PER MONTH. (USUALLY USED IN PLACE OF MONTHLY PAY HOURS)
57	RUBRHR[I]	PAY HOURS INCLUDED SYNTHETIC FLIGHT TIME (INPUT; MAX BID/GUAR BID/GUAR RES/OTHER/OT
58	SOBASE	<u>MONTHLY</u> BASE PAY FOR SECOND OFFICERS BY YEAR OF SERVICE
59	SODAYRATE	HOURLY RATE OF FLIGHT PAY OR THE DAY RATE OF FLIGHT PAY FOR SECOND OFFICERS
60	SODFLAT	FLAT SALARY FOR FIRST YEAR DOMESTICE SECOND OFFICERS.
61	SODHRS[I]	DOMESTIC SECOND OFFICER HOURS (INPUT; MAX BID/GUAR BID/GUAR RES/OTHER/OTHER)
62	SOIHRS[I]	INTERNATIONAL SECOND OFFICER HOURS (INPUT; MAX BID/GUAR BID/GUAR RES/OTHER/OTHER)
63	SOIFLAT	FLAT SALARY FOR FIRST YEAR INTERNATIONAL SECOND OFFICERS
64	SOINTL	INTERNATIONAL OVERRIDE PAY FOR SECOND OFFICERS
65	SOLONG	HOURLY BASE/LONGEVITY PAY FOR SECOND OFFICERS BY YEAR OF SERVICE
66	SONAV	NAVIGATION (DOPPLER/INS) OVERRIDE PAY FOR SECOND OFFICERS
67	SONITERATE	NIGHT RATE OF FLIGHT PAY FOR SECOND OFFICERS
68	SOODP	OPERATIONAL DUTY PAY FOR SECOND OFFICERS
69	SOODPHRS[I]	OPERATIONAL DUTY PAY HOURS (INPUT; MAX BID/GUAR BID/GUAR RES/OTHER/OTHER)
70	SOOUTPCT	THE PERCENT OF GOING-OUT RATES APPLICABLE TO SECOND OFFICERS AT GIVEN EFF. DATE
71	SOPCT	THE PERCENTAGES OF CAPTAIN/FIRST OFFICER PAY APPLICABLE TO SECOND OFFICERS
72	SODER	1=PCT TOTAL CAPT,2=CAPT FLT,3=TOTAL F/O,4=F/O FLT, AND 5=NONE OF ABOVE
73	TRIPRATE	THE RATE OF PAY PER TRIP. (USUALLY USED IN CONJUNCTION WITH THE ACRONYM TRIPNOS[I])
74	SPEED	NEGOTIATED AIRCRAFT SPEED
75	WGT	NEGOTIATED AIRCRAFT WEIGHT IN <u>THOUSANDS</u> OF POUNDS
76	WGTLIMIT	THE 1ST AIRCRAFT WEIGHT UP TO WHICH THE 1ST WEIGHT RATE OF PAY IS APPLICABLE
77	WGTLIMIT2	THE 2ND AIRCRAFT WEIGHT UP TO WHICH THE 2ND WEIGHT RATE OF PAY IS APPLICABLE
78	WGTLIMIT3	THE 3RD AIRCRAFT WEIGHT UP TO WHICH THE 3RD WEIGHT RATE OF PAY IS APPLICABLE.
79	WGTLIMIT4	THE 4TH AIRCRAFT WEIGHT UP TO WHICH THE 4TH WIEGHT RATE OF PAY IS APPLICABLE
80	WGTRATE	AIRCRAFT WEIGHT RATE OF PAY (IN CENTS...3 CENTS=.03)
81	WGTRATE2	THE 2ND WEIGHT RATE OF PAY (IN CENTS...3 CENTS=.03) IF MORE THAN ONE RATE EXISTS
82	WGTRATE3	THE 3RD WEIGHT RATE OF PAY (IN CENTS...3 CENTS=.03) IF MORE THAN ONE RATE EXISTS
83	WGTRATE4	THE 4TH WEIGHT RATE OF PAY (IN CENTS...3 CENTS=.03) IF MORE THAN ONE RATE EXISTS
84	XFIELDA	THIS IS A DUMMY VARIABLE WHICH CAN BE USED IN INTERMEDIATE CALCULATIONS
85	XFIELDB	THIS IS A DUMMY VARIABLE WHICH CAN BE USED IN INTERMEDIATE CALCULATIONS
86	XFIELDC	THIS IS A DUMMY VARIABLE WHICH CAN BE USED IN INTERMEDIATE CALCULATIONS
87	XFIELDD	THIS IS A DUMMY VARIABLE WHICH CAN BE USED IN INTERMEDAITE CALCULATIONS
88	XFIELDE	THIS IS A DUMMY VARIABLE WHICH CAN BE USED IN INTERMEDIATE CALCULATIONS
89	SOINC	ENTER 1 IF 2ND OFFICERS FLY GIVEN EQUIPMENT; ENTER 0 IF THEY DO NOT.
90	PROBDAYRATE	HOURLY RATE OF FLYING PER DAY FOR PROBATIONARY (FIRST YEAR) PILOTS
91	PROBNITERATE	HOURLY RATE OF FLYING PER NIGHT FOR PROBATIONARY (FIRST YEAR) PILOTS

Figure 2 (Part 2)



```
CONTRACT KEY: O2A,0878,1,0,1
UNION: ALPA
AMENDABLE DATE: 013181
EQUIPMENT TYPE: FH-27B
EFFECTIVE DATE: 03 01 79

DAYRATE       : 33.9  33.9  33.9  33.9  33.9  33.9  33.9  33.9  33.9  33.9  33.9  33.9
NITERATE      : 36.9  36.9  36.9  36.9  36.9  36.9  36.9  36.9  36.9  36.9  36.9  36.9
SPEED         : 300  300  300  300  300  300  300  300  300  300  300  300
WGT           : 45  45  45  45  45  45  45  45  45  45  45  45
MILERATE      : 0.03  0.03  0.03  0.03  0.03  0.03  0.03  0.03  0.03  0.03  0.03  0.03
WGTRATE       : 0.03  0.03  0.03  0.03  0.03  0.03  0.03  0.03  0.03  0.03  0.03  0.03
BASE          : 245  265  285  305  325  345  365  385  405  425  445  465
FOPCT         : 0  0.44  0.47  0.55  0.59  0.6  0.61  0.62  0.62  0.62  0.62  0.62
CAINTL        : 3.5  3.5  3.5  3.5  3.5  3.5  3.5  3.5  3.5  3.5  3.5  3.5
ENTITY        : 3
FODER         : 2
DAYRATIO[I]   : 0.5  0.5  0.5  0.5  0.5                ------------------
CADFLAT       : 0                                       CAD/FOD/SOD/CAI/FOI/SOI
CAIFLAT       : 0                                       MAX BID:
FODFLAT       : 900                                       4125  2734     0  4405  2908     0
FOIFLAT       : 900                                     GUAR BID:
CADHRS[I]     : 80  69  69  75  85                        3622  2422     0  3863  2572     0
CAIHRS[I]     : 80  69  69  75  85                      GUAR RES:
FODHRS[I]     : 80  69  69  75  85                        3622  2422     0  3863  2572     0
FOIHRS[I]     : 80  69  69  75  85
SOINC         : 0  0  0  0  0  0  0  0  0  0  0  0

FORMULA
CADEND←BASE+CADHRS[I]×CADFLT←(DAYRATE×DAYRATIO[I])+(NITERATE×NITERATIO[I])+(SPEED×MILERATE)+(WGT×WGTRATE)
CAIEND←BASE+CAIHRS[I]×CAIFLT←CADFLT+CAINTL
FODEND←BASE+FODHRS[I]×FODFLT←FOPCT×CADFLT
FOIEND←BASE+FOIHRS[I]×FOIFLT←FOPCT×CAIFLT
```

Figure 3

CORPORATE DECISION-MAKING & DATA BASE APPLICATIONS

Eike W. Kaiser
Rowntree Mackintosh Canada Ltd.
Toronto, Canada

Introduction

Most decisions made in business are only arrived at after some analysis of historical facts. The analysis may be as simple as just reflecting on one's own experience, or so detailed that mountains of paper are created in the process. The degree to which either of these extremes applies to any particular analysis will depend on the time-frame imposed, the amount of data involved, the methods available for manipulating it, and the expected importance of the answer. Very often it is the time-frame and the methods of manipulation which are the deciding factors, since they will influence the amount of data that can be considered and therefore the quality and relevance of the result.

The time-frame for information and decision-making requirements can seldom be changed, so our prime concern today is with methods of manipulation. Businesses today seem still to provide an inadequate level of computer support for this aspect of the decision-making process. Their computers work around the clock producing data which should be a valuable resource to the company, and which was once thought to be the answer to these information problems. Now we realize, though, that business tends to have stopped one step short, and addressed only the first stage of the problem which is the collection of data. In this process they provide some comparisons by which to evaluate it, but these are generally useful only for identifying highlights and not for accurately defining the degree, duration, causes or even importance of the observed trends. Further analyses seem always to be required to identify these features, and these analyses tend to be done manually. Such an approach has many drawbacks which are not consistent with the need for effective and timely decision-making.

In the many rewrites that this presentation has undergone, it has been difficult to stay with the original objective of just describing our own solution to these problems. The relevance of any such solution is difficult to understand without adequate definition of the problem and its causes and implications. These issues are all a part of the solution and they must all receive some mention. While I intend to tackle this comprehensive overview I hope that the descriptive section — which is likely to provide the greatest stimulus to others and the greatest feedback for ourselves — has not suffered as a result.

Our Environment

Our examples and discussion will be taken from a marketing department's point of view. This happens to be the area that I am involved in, and it is one which I know tends to be poorly serviced by computer facilities. I'll cite proof of this contention in a few minutes, though you'll still need to judge for yourselves whether the examples used later would be considered as unusual or as improved levels of service in most companies. To my knowledge they would be, but little factual evidence for or against this view has been found. Marketers do have very large amounts of data to work with though, and they must apply this in defining brand and corporate strategies. Our own data base, which has only existed for two years in a computerized form, is structured as shown in Exhibit I.

This data base is divided into five major areas, with each area having it's own subdivisions. Because of the inter-relationship of many of the measures represented here, our internal sales data has purposely been shown in a central position. The other factors that jointly influence these results have been shown as peripheral data. For example, our own data is a part of the confectionery industry reports produced by Statistics Canada and shown as industry statistics, and also a part of the pooled shipments and retail audits that are shown under trade data. In turn, all of these measures are influenced by population and economic factors, which are shown here as general information, and by consumer traits which we attempt to identify through private research.

To illustrate the volume of data involved here, if we were to print it out in one long line at ten characters per inch then it would stretch for a distance of about twenty miles. I have only one assistant to help in turning all of this data into information that is useful to our marketing staff, our sales team, or any others on our management team. Our first problem in a non-computerized environment would be that of just finding the data that is required. Even in a well organized filing system, which in this analogy would be closely equivalent to placing ourselves at the five-mile and fifteen-mile positions along this data path, this task can still be monumental. This analogy is validated when one realizes that our pooled shipment data, which was listed in Exhibit I, theoretically occupies 260 two-inch binders in our archives. A minimum of forty of these binders would have to be found to conduct a ten-year analysis of any feature. Their data would all have to be copied to working papers of some sort, and only then would we finally have reached the same point that we now start from in a computerized environment. In real life these numerous volumes occupy only a small part of the surface of one small computer disc.

The Computer's Role

The computer virtually eliminates these problems for us. It might be true that because of the computer we are doing more work than necessary, and that the ten-year analysis referred to here is an exploitation of the computer facility available to us. Some later examples will hopefully show that such long-term analyses are very important, but I will also admit to doing more work than is absolutely required. In fact, I'll plead guilty to exploiting the computer facility too. Here it must be realized that a major part of our job is just to correctly identify the question that we are working on. The question "What is wrong with Brand X?" may be legitimate, but it can only be answered after many other questions are examined: Wrong?... Compared to what? In which regions? Since when? How have other brands performed? These are the questions that gradually lead to the final answer.

So far, it has been assumed that none of you would be confused or afraid when confronted with only a question and a computer keyboard. However, we must be careful not to make this assumption about most managers in business. Many of them are totally convinced that only a specialist can survive such an ordeal, and then only after considerable training in incomprehensible languages that somehow seem to be less than human. Few of them have had any opportunity to learn differently, and this is probably the greatest reason for the lack of computer applications such as those we are discussing.

In part, these managers are actually quite right in their assessment. The problems we are considering here tend to be constantly different from one another, and frequently even the nature of the final answer is only poorly understood until much of the work has already been done. To approach such problems with conventional programming languages probably would be more cumbersome than helpful. Since most managers have quite enough experience in waiting for in-house applications to be programmed, we can at least understand their tendency to rule out direct computer access as being of any potential benefit. Fortunately, little of this experience is relevant to an APL environment.

Unfortunately, without management recognition of the potential in this area, and thus without active management support for computerization, there tends to be little impetus for change in this area.

Getting Started

Even though much of APL will take some study to master there is also a great deal that a novice can master with only superficial instruction. This, in fact, is how our installation started only two years ago. We had selected I.P. Sharp to supply a prompted, English-language version of statistical programs through their time-sharing service. The initial budget for this project was about $1500 per year, and we viewed this project as a priority. To help in structuring our data for these programs, I attended Sharp's three-day introductory APL seminar, and I returned from it full of fire and enthusiasm. Within three weeks my computer budget had increased fifteen-fold, and since that time it has doubled again. That budget is devoted strictly to our marketing information and decision-making requirements, and it has no day-to-day operational applications. The priority project that all of this started with has sat virtually untouched for these two years, while the decision-making applications being discussed here were developed. Interestingly, we know that the clerical staff which could be hired with the same budget that we use for these APL applications could not hope to match the productivity that we now achieve by the use of the computer. It is also interesting to note that this level of service was never expected or even missed prior to our use of APL. It would be sorely missed now though.

The fact that my skyrocketing budget and my lost time on priority projects have not cost me my job should give some proof of this point. The budget, incidentally, does not have to skyrocket. Once it gets to be a little bit heavy there are excellent alternatives available to lighten it up again. We are currently planning to bring APL onto our in-house computer, and this will replace substantial time-sharing costs with a very minimal amount of new but constant overhead. Since this will not happen for about a year and a half when a new computer arrives, we are also considering the purchase of a $7000 minicomputer with a very effective APL which will take much of the sting out of our time-sharing expenditures in the interim.

Another aspect of the manual approach to analyses needs consideration, namely, who should be conducting them. In many companies, it is the assistant to whoever it is that happens to have a question, or whoever it is that has a question asked of them, that ends up doing the analysis. Few management level personnel have the time to pour over masses of numbers, and often, neither they nor their staff will have the skills to thoroughly evaluate what they find there. Our two-man department — although we do not appreciate all of the intricacies of the marketing or sales departments — are better equipped to understand their problems than most programmers or clerical assistants, better equipped to understand computers than most marketers or their assistants, and very well qualified in numeric techniques. This combination gives reasonable assurance that we will find the answers to any question if this is at all possible.

A final point to consider is timeliness. Many of our analyses involve literally thousands of calculations. These would include applications such as simple listings of the top twenty-five brands in our market in every region of the country. This little gem is especially appreciated by our sales department, but many of these jobs could not possibly be completed within the time-frame of the information requirements that prompt them. Here computers are the only viable solution, and the combination of qualified staff, a comprehensive data base, and the powerful APL language make this solution effective for us.

Some Proof of Inadequacy of Computer Support

I've belaboured the problem and the solution enough now, and I should provide adequate proof of my various claims and contentions. To start this an introduction to one of our research reports is appropriate. This example comes from our trade data base, which is simply data that all of our major competitors have access to. We'll start with a retail audit produced by A. C. Nielsen. The seven inch stack of papers they give us is updated every two months, complete with almost 400,000 new numbers. This report describes the retail environment faced by our products across Canada. Among other things it provides measures of market share which we expect will mirror consumer behavior, of the effects of the various retail influences reported on, and of the effects of factors such as advertising and promotions.

Neilsen have over 400 contracts for this service in Canada, with about 140 different clients. Admittedly, not all of the reports are as voluminous as ours for candy bars, but still their cumulative effect over several years can be ominous. The interesting fact about all of this is that only six of these 140 clients have requested computer tapes of this data. All of them must pay a very substantial fee for this service, so the data must be important, and yet fewer than five per cent of them have computer facilities for examining it. Ninety-five per cent attempt to cope with this manually, and to integrate it with their other data. The sample which this little survey is based on, is obviously biased towards the larger and more successful companies who can afford the data, and who realize the importance of knowing what happens to their products after they leave the shipping dock. If companies who do not purchase Nielsen data were included in this little survey then I'm sure that the five per cent we arrived at here would have been even smaller. This proves at least part of my earlier claim that most marketers and decision-makers tend to receive less computer support than could be justified by their needs.

Applications with Retail Audits

Especially now that A. C. Nielsen have developed their "electronic report book", as they call it, this low incidence of computer usage is difficult to understand. This service provides Nielsen's clients with a computer tape of all of their data combined with a set of English-like programs that can be used to manipulate that data. There is little left to do except to run this tape into their own computer or into a time-sharing facility if this is preferred. There is absolutely no programming to learn. All that is needed is a general understanding of how the whole service is organized. Even though all of the work behind computerization of this application has been done by Nielsen only a small minority of manufacturers are taking advantage of it. The pricing of the service may partly explain this, but but a combination of the unfamiliarity and awe with which marketers view the computer are probably more important factors. Even if price were the major factor, this should not be a deterrent. With a language like APL, the programming required for such a system can be kept very simple. That comment is made with I.P. Sharp's file system in mind, and it may need qualification in other installations. Still, this in-house programming should save the cost of renting Nielsen's programs. It should also allow a more effective system to be developed than what Nielsen can currently provide. It can be designed to meet the specific requirements of the user as in the example provided in Appendix I. This looks massive at first, and I can only explain a little of it, but it will be well worth some study by any Nielsen clients who are considering computerization of this data. The programming effort that went into this was really quite small though. Since even this small programming task may not appeal to all users, the service Nielsen provide can still meet a very important business need. Keep watching Nielsen here, because their product is certain to improve, and talk to them in detail about getting the most from this service.

(The above comments are not intended as a slight against what Nielsen have done. Their efforts can make a great contribution to improving the effectiveness and efficiency of any company's marketing department where this work is still being done manually. At the same time though, the Nielsen version usually requires a new command from the user for every single type of data that is to be printed or calculated. As you can see here, our approach requires only one command to generate a complete and comprehensive summary. Individual measures can also be extracted, but we find that the full report lends itself much more effectively to rapid and thorough analysis. This approach will always present more measures than may be needed in any evaluation, but the economics of using computers make this a smaller drain on human and capital resources than would result if every measure required were to be requested separately. On this basis, we judge Nielsen's effort to be somewhat uninspired, and something that they and other manufacturers can easily improve upon.)

Manually, a comprehensive report such as this would take hours to compile. Our program provides it within minutes, for less than $8.00, and for any brand within any region that we wish to specify. Many of the measures that we report here are copied directly from the data base, while others are calculated from it. Usually our analyst needs only to inspect this one page to immediately answer any questions that arise. The time between the original asking of a question and a definitive answer usually does not give the questioner time to return to his office. The output of the program also leaves little chance that any salient features might have been overlooked. Our approach to using this document is shown by the various tracks we have left across this appendix to highlight some of the facts and relationships that seem noteworthy. If any additional analysis is required, then all of the output of this program remains available to the analyst for further calculations. For example, we may want to examine the statistical significance of certain out-of-stock measures, or distribution measures, or of their impact on sales share. We can directly take the result provided by this program and use it as input to whatever set of programs next seem appropriate. Obviously, the report shown in this appendix would not be circulated for general information, but would be used almost exclusively by ourselves, to ensure that the summary report that we finally issue is thorough and complete.

Remember that the program which generated this report was written by a novice

APL'er. That doesn't need to restrict its effectiveness at all, but might make running it slightly more expensive than if it had been expertly programmed. This program has evolved a great deal since it's first writing, to match our needs, our growing understanding of the data, and our growing expertise in programming. As a result, the worst of these minor inefficiencies probably disappear with time in any case.

Our computer is also used in comparing Nielsen's report volumes against our internal sales information, even though the timing of the two reports is very different. Similar tasks are done in converting economic and pricing measures from Statistics Canada's monthly basis to our own calendar timing. All of the conversions required to change over from one timing to the other are allowed for in our programs for this purpose, and the whole task takes only seconds to run. These programs were developed in less than an afternoon's worth of work. Except for updating with each year's new calendar, this particular task will never take up any more of our time or attention again. The efficiency that this provides in eliminating detailed and time-consuming clerical jobs is tremendous.

Applications with Sales Data

Another part of our trade data base is our pooled shipments data. Confectionery is sold through a very complex distribution system and is probably available in more types of stores than virtually any other commodity. Because of this, the research work done by companies like A. C. Neilsen has difficulty in representing volumes accurately, and they may even present share relationships that are widely off the mark. To overcome this, the major candy bar manufacturers have shared their shipment information with one another through an outside computer bureau. From this data, we know exactly the volumes shipped for each of about 200 different products in all six of the regions into which we divide the country. Ten years of this history, including totals for separate companies and market segments, are available to us. Unfortunately, except for the data base we created for our own use, this service exists only as hard-copy printouts. Competitors still have not followed our lead in computerizing this report.

As one example of this data base, we could consider its use in examining the effect of provincial sales taxes on candy bars. At different times over this ten year history, various provinces have imposed or repealed sales taxes on candy, and these changes have influenced our sales volumes. About a year ago one of our vice-presidents came over, saying that he was about to see one of these provincial governments to discuss their recent introduction of sales tax. He needed some facts and he was leaving in half an hour! With the 260 volumes of binders that this data was stored in, we would have had little chance of helping him. With the computer we had time to sit and discuss the results with him before he left. I've recreated those results here as Exhibit II, using about 10 minutes of time and $12.00 of computer costs. The 1977 decline in the upper line here was tax related, as was its reversal in 1978, and the 1979/80 decline in the lower line is also tax related. This gave our V. P. some very convincing but still simple and thorough documentation around which to focus his discussions.

As an aside, I should mention that the graph in Exhibit II was produced on the computer terminal in our office, using the I.P. Sharp "3 Plot" graphing package. Graphic support like this is essential to our function because it permits rapid evaluation of data, through which we develop hypotheses about the questions being considered. No matter how we may test out any hypotheses, we invariably also use graphs in our final documented answer to most questions. The Sharp facility meets these needs extremely well.

An unusual feature of the candy bar market is that products developed in the last 10 and even 20 years account for only a small portion of total sales. This type of information is fairly well documented for our planning purposes and for monitoring of our own, as well as competitive new introductions. The awareness that many industries are achieving the majority of their current sales from such new products, and the search for competitive share gains within our industry, constantly raise questions about new products for us. Exhibit III summarizes our industry's history in this area very nicely. It shows the combined share at any point in time of all the new products that have been introduced in the last ten years. The rise and fall of individual brands within this total is clearly seen. This result was obtained in about 25 minutes — most of it for plotting the rather complicated graph — and about $30.00 in computer costs. Any manual approach to this same request would have been much more expensive. In fact, in a manual environment this and other tasks might never be undertaken, and thus our understanding of the problems and of the market as a whole would remain correspondingly less complete.

Other applications for this industry shipments data are extremely varied and depend on our needs at any time. For example, not that I like to remind you, what we call a regular size bar used to cost ten cents, and you used to have a choice of a five cent, fifteen cent or twenty-five cent version of most of the popular brands. Now the regular size costs thirty-five cents. Also, at least until very recently, your choice in selecting a product was reduced to only two sizes, this regular one and the much larger family bar at about seventy-five cents. For our short-term planning these price changes cause concern as to whether the amount of erosion is similar for all manufacturers, market sectors and regions. For our long term-planning, we want to be able to predict any recovery from this erosion, and the likely effect of any future price increases that we might be able to foresee. For this purpose we have had some success in building statistical models of our market using the industry shipments data. On the choice front, since other sizes were successful in the past we have considered re-introducing them, and our ability to quickly examine historical data on their performance has been very much of a help in evaluating this opportunity.

These examples hopefully illustrate several points from the discussion earlier, namely:

- speed of access to data
- speed of execution
- ability to handle non-routine requirements
- extended analytical abilities
- ease of mastering APL programming

Maximizing the Benefits

The degree to which these benefits will be realized depends partly on the language being used — and we still find APL to be totally suitable here — and on how comprehensive the data base is. We often need to integrate various measures such as the Consumer Price Index with our trade data. At other times we need to integrate it with industry data from Statistics Canada and then to add population and other research data into the mixture as well. Such requirements tend to relate to our budgetting process but they do also come up on other occassions. As a result, all of these small and manageable measures are also retained in our data base along with the large and unwieldy ones. This lets us take full advantage of the computer's abilities, and avoids any unnecessary reliance on slower manual techniques. This is what I refer to as a comprehensive data base. If any measures that influence the market are not represented in the data base, then they are liable to be ignored in an analysis or they will cause unnecessary delays in its execution. Such an omission would therefore minimize the benefits that can be realized from computerization.

We have also found that new and important uses for these ancilliary measures will develop once they can be manipulated so easily for any purpose. An example from a very small part of our data base will prove useful in clarifying this. This whole set of data consists of only about 400 numbers which summarize Statistics Canada's reports on the confectionery market since 1963. For each year, it shows unit and dollar sales for such major segments as gum, sugar candy and chocolate. The size of this data obviously does not justify computerizing it. Since it has been computerized though, we have examined trends such as the dollars per pound implied in the data, and used this to identify, comfirm and correct discrepancies in the published numbers. We have also used it to examine the share trend of chocolate within the total market. That trend is shown in Exhibit IV, which actually turned out not to be very enlightening. It does show the historical facts, but there is no obviously relevant use for these facts, other than as information. With access to the computer we took this analysis one stage further though, and plotted this chocolate share as a function of chocolate's price relative to other confectionery. This result is shown in Exhibit V, which is seen to be much more informative. In the form shown here, it suggests a loose relationship that explains chocolate's share trend and provides guidance to our planning for both volumes and prices. The volume levels agreed to in one stage of our budgetting process provide us with a direct estimate of chocolate poundage, while the price trends estimated in another stage now allow us to prepare a second estimate of this poundage as well. If these two estimates match up well, then our confidence in the forecast increases. If they do not then we have an opportunity to re-examine our assumptions and to determine whether we have applied them consistently in our two approaches to this task. We feel that this has resulted in a more thorough and rational approach to our budgetting than we had used previously.

This type of work is obviously the prelude to the more rigorous statistical modelling of our industry which is the priority project that was mentioned earlier as having been placed in abeyance while our work on these analytical and decision-oriented projects was developed. In this modelling work we have already refined the relationship discussed in the previous example, so that it does provide a better guide to the budgetting application than would be suggested by the simple graph used to represent it here. The requirements of this type of work make the availability of a computer essential. Since interpretation and refinement of these models of the marketer's data are constantly required, this only argues more strongly for better and more direct and flexible computer support in this area.

One final example deserves mention. This is a particular favourite of mine, because it is almost eloquent in its combination of the pitfalls of manual analyses. It is ironical too, because part of the analysis was done using our data base. Here our analyst was so swayed by the evidence that had already been collected that even he failed to identify the real factors that were at work. What happened was that one of our regions seemed to be having some difficulty in the sales of one of our products. This was shown through our run-of-the-mill internal sales reports, where this region's performance compared to a year earlier was several per cent lower than any other region's.

Several items of information had been gathered manually, and showed that this problem had been developing for some time. Retail audit data was used to confirm the finding, and several factors had even been identified as the cause of this weakness. By the time this problem arrived on our desk, the evidence seemed overwhelming and corrective measures had already been defined. Our analyst basically just repeated the manual procedures and confirmed their results. In reviewing his work, I was surprised to see that only the immediately preceding year had been considered in this whole evaluation. Based on the facts already accumulated and the briefing that he had been given this appeared to be adequate, but I just wasn't at ease with it. The ten year trend for this product and region are shown in Exhibit VI. This makes it clear that the year-ago data was the unusual feature here, and that the current sales results still represented a very strong position for this region. All the work identifying its weakness as compared to a year earlier was indeed correct, but it was also irrelevant since the current results remained stronger than in any other part of our history.

Numbers have a funny way of being misleading. In this case, they would have led us into unnecessary corrective action that would have been expensive financially, as well as in terms of morale. Identifying this error took only a few minutes of work, although convincing others about it did take a bit longer. Thanks to the availability of a comprehensive data base and a powerful programming language we seldom fall into these pitfalls anymore.

Summary

The analysis of corporate data still tends to be treated much more as an art than as a science. Frequently, there are neither staff nor computer facilities allocated specifically to this function. Questions tend to be something that managers almost stumble onto in the course of their other duties. Answers tend to be something that their assistants attempt to compile after a quick briefing, a laborious search for data, an admirable attempt at wearing out their calculator, and generally little further thought.

This approach has never really been adequate, but today it is also no longer necessary. Sufficient expertise with high level programming languages such as APL can be developed with so little training that the computer has become as practical a tool for business analysis as the calculator is to simple clerical functions. The cost of computers has also fallen so sharply that they are now an efficient and affordable tool for almost any size of company.

Any initial investment required for computerization of these functions will be quickly recovered by the speed, accuracy, thoroughness, and efficiency which computers permit in this area. With increasingly discerning consumers, increasingly demanding governments, and increasingly aggressive competitors, this function will be more and more in need of the attention for which it is already overdue.

TITLE: [9 ; 1 29 ; 2 3 4 5 8 9 10 11 14 15 16 17]

BRAND "A" REGION "B"

JULY - OCT LEFT OUT TO PERMIT SHOWING MORE HISTORICAL DATA.

	N/D	J/F	M/A	M/J *	N/D	J/F	M/A	M/J *	N/D	J/F	M/A	M/J	
RGN •/•NAT: INDUST	12.41	7.82	9.74	10.11	13.23	7.16	8.46	9.11	15.17	10.09	10.13	11.79	SEASONAL PATTERN
BRAND	22.82	7.06	9.79	12.37	19.54	7.46	8.53	7.05	17.61	14.49	9.36	9.19	BRAND HAS STRONGER PEAK
SHARE: MAT UNITS	~~1.00~~ N/A	~~1.00~~	~~1.00~~ N/A	~~1.00~~	12.66	12.43	12.04	11.13	8.34	9.19	9.17	9.26	IMPACT OF WEAK XMAS
MAT DOLLARS	~~1.00~~	~~1.00~~	~~1.00~~	~~1.00~~	15.67	15.46	14.85	13.81	10.87	12.18	12.17	12.17	
MAT POUNDS	~~1.00~~	~~1.00~~	~~1.00~~	~~1.00~~	13.67	13.58	13.23	12.31	9.56	10.52	10.42	10.40	
1/2 YR POUNDS	1.00	14.37	14.42	13.30	13.90	12.06	11.39	9.83	9.40	10.18	10.26	11.73	PART OF LONGER TREND
OPEN INVENT	9.04	9.47	12.48	10.05	5.31	7.21	9.68	6.83	2.08	7.37	7.34	7.78	* PROBLEM : WEAK RETAIL INVENTORIES
+ PURCHASES	16.51	18.35	11.30	31.50	13.55	13.23	12.14	18.29	11.86	16.57	15.21	12.94	HEAVY RELIANCE ON CURRENT PURCHASES TO GENERATE SALES ... LOW RISK, BUT LOW SALES TOO
= AVAILABILITY	13.79	11.22	12.02	17.61	10.47	9.62	8.98	9.53	7.99	10.79	8.81	8.70	
- SALES	16.32	10.07	14.63	17.66	12.07	9.58	11.55	8.34	8.38	13.93	10.51	8.59	
\ = CLOSE INVENTORY	9.47	12.48	10.05	17.57	7.21	9.68	6.83	10.76	7.37	7.34	7.78	8.80	
SELL OUT RATE	74.60	46.70	52.31	45.67	77.24	53.19	58.73	44.39	64.83	67.53	45.06	46.74	
REPLACEMENT RATE	102.42	122.56	67.12	131.94	105.11	87.39	104.42	114.32	137.22	70.42	130.52	63.11	
SELL OUT INDEX	1.18	0.90	1.22	1.00	1.15	0.99	1.29	0.87	1.05	1.29	1.19	0.99	PRODUCT HAS HIGH TURN-OVER WHEN AVAILABLE
REPLACEMENT INDEX	1.01	1.82	0.77	1.78	1.12	1.38	1.05	2.19	1.41	1.19	1.45	1.51	
DOLLARS PER POUND	4.45	4.48	4.48	4.27	4.46	4.83	4.65	4.65	5.25	5.80	5.53	5.55	ALWAYS AT PRICE DISADVANT.
IND $/LB.	3.76	3.91	3.63	3.73	3.91	4.27	4.26	4.13	4.47	4.97	4.69	4.68	
POUNDS/UNIT: BRAND	0.86	0.70	0.75	0.78	0.86	0.69	0.79	0.77	0.90	0.76	0.74	0.70	
INDUST	0.79	0.77	0.73	0.72	0.75	0.71	0.73	0.68	0.72	0.74	0.72	0.70	
DEAL SHARE: POUNDS					1.00						1.00	1.00	
STORE IMPORTANCE	130	126	120	118	114	121	110	135	103	111	116	112	SOLD MOSTLY BY HIGH-VOLUME STORES
S C D DISTRIBUTION	37	35	35	38	35	42	31	20	30	36	32	33	
STORE O/S		10	6	11	2	8	15	3		6	5		
SHELF O/S		10	6	11	2	8	15	3		6	5	1	BUT ONLY A FEW.... - REACHABLE - CAN SUPPORT BETTER INVENTORY - EXPLAIN OUR TURN-OVER STORY
DISPLAY													
LOCAL ADVTG													
MENTIONS													
AL COM DISTRIBUTION	48	44	42	45	40	51	34	27	31	40	37	37	
STORE O/S		7	2	8	3	14	8	7		4	7		
SHELF O/S		7	2	8	3	14	8	7		4	7	1	
DISPLAY													
LOCAL ADVTG	NEVER PROMOTED ... WHY?												
MENTIONS													
ADVERTISING: IND $													
BRAND SHARE PD	100	100	100	100	100	100	100	100	100	100	100	100	
MAT	1	1	1	1	1	1	1	1	1	1	1	1	
BRAND •/• TV	1	1	1	1	1	1	1	1	1	1	1	1	
•/• RADIO	1	1	1	1	1	1	1	1	1	1	1	1	
•/• PRESS	1	1	1	1	1	1	1	1	1	1	1	1	
OPENING INVENT (LBS)	1203	1280	1194	876	782	929	879	525	370	1303	949	923	RAW DATA FOR FURTHER CALCULATIONS IF REQ'D
PURCHASES (LBS)	3851	1282	645	1330	3313	873	780	543	3296	1390	988	385	
SALES (LBS)	3760	1046	961	1008	3152	999	747	475	2402	1974	757	610	
CLOSE INVENT (LBS)	1280	1194	876	1199	929	879	525	595	1303	949	923	695	
VOL EX IND TREND/SEA					1166	925	1114	804	846	1493	1139	951	DE-SEASONALIZED SALES
VOL PER AC DIST PT	78	24	23	22	79	20	22	18	77	49	20	16	
-EX IND TREND/SEA					29	18	33	30	27	37	31	26	

BIG STORES OUT-OF-STOCK

SMALL STORES OUT-OF-STOCK

LOST RE-ORDERS (PRICE OVER $4.50 ???)

****** MISSING NUMBERS INDICATE ZERO SHARE; '1.00' INDICATES BOTH BRAND AND INDUSTRY WERE ZERO (EXCEPT IN SPECIAL CASES EG: POUNDS PER UNIT).

```
==========================================================
|             TABLE I:  CONFECTIONERY DATA                |
|             ~~~~~~~~~~~~~~~~~~~~~~~~~~~~                |
|                                :                        |
|     INDUSTRY STATISTICS        :   GENERAL DATA         |
|     -SALES VOLUMES             :   -POPULATION          |
|     -PRODUCTION                :   -DEMOGRAPHICS        |
|     -INVENTORIES               :   -ECONOMIC INDICATORS |
|     -EMPLOYMENT AND COSTS:                              |
|                                :                        |
|            ----------------------------                 |
|            |                          |                 |
|            |      INTERNAL DATA       |                 |
|............|DETAILED SALES RECORDS    |.................|
|            |      PRICING DATA        |                 |
|            |      DEALING PERIODS     |                 |
|            |                          |                 |
|            ----------------------------                 |
|                               :                         |
|     TRADE DATA                :    PRIVATE RESEARCH     |
|     -RETAIL AUDITS            :    -MARKET STUDIES      |
|     -POOLED SHIPMENTS         :    -TRACKING STUDIES    |
|                               :    -BRAND PROFILES      |
|                               :                         |
==========================================================
```

Exhibit I

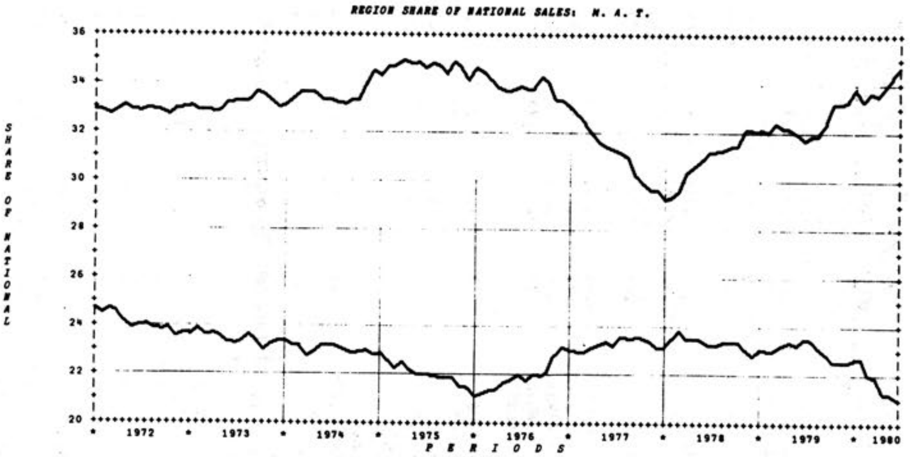


Exhibit II

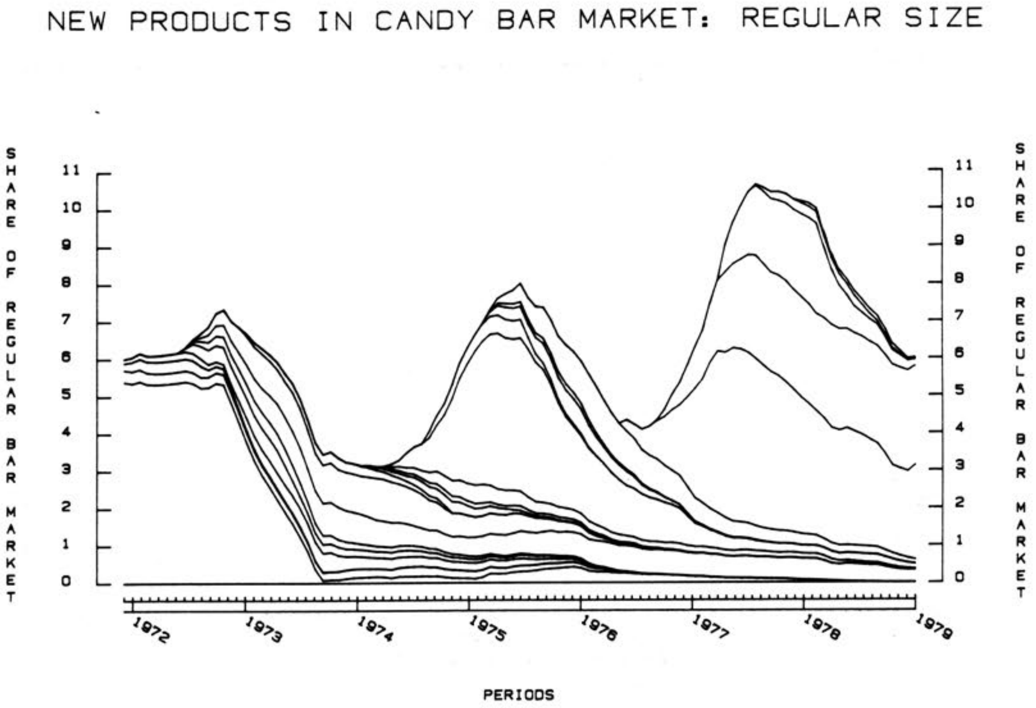


Exhibit III

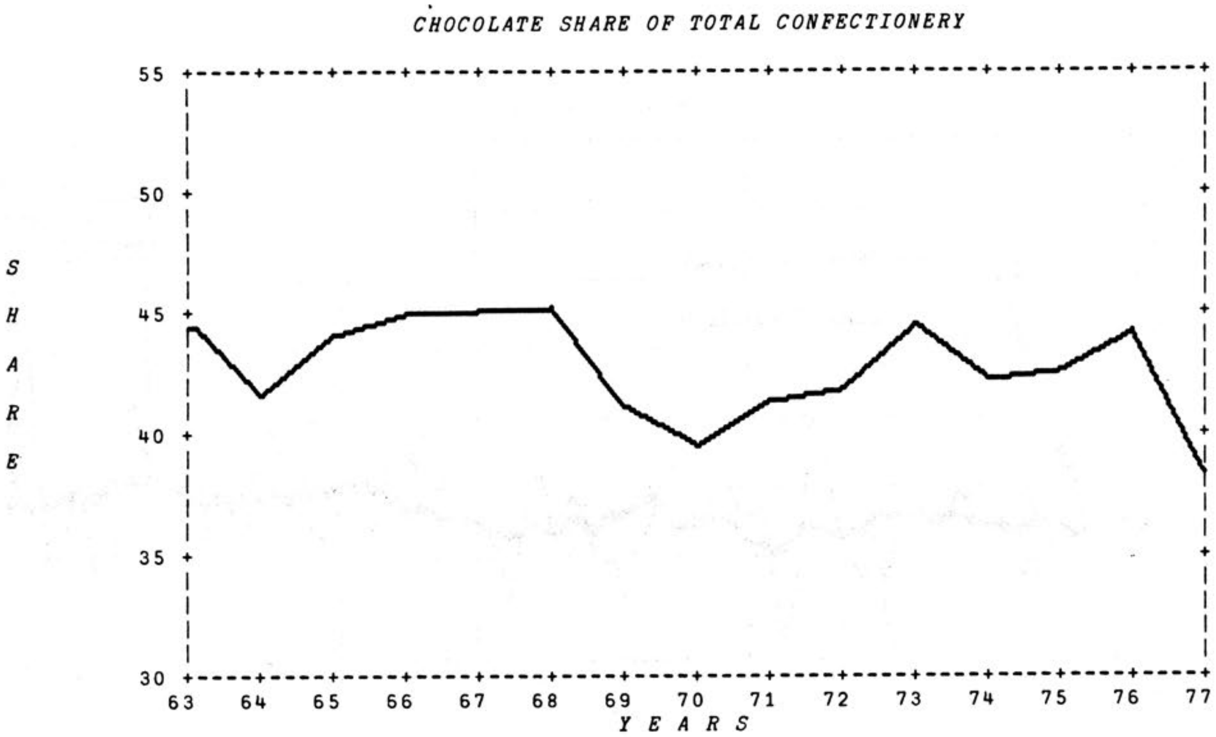


Exhibit IV

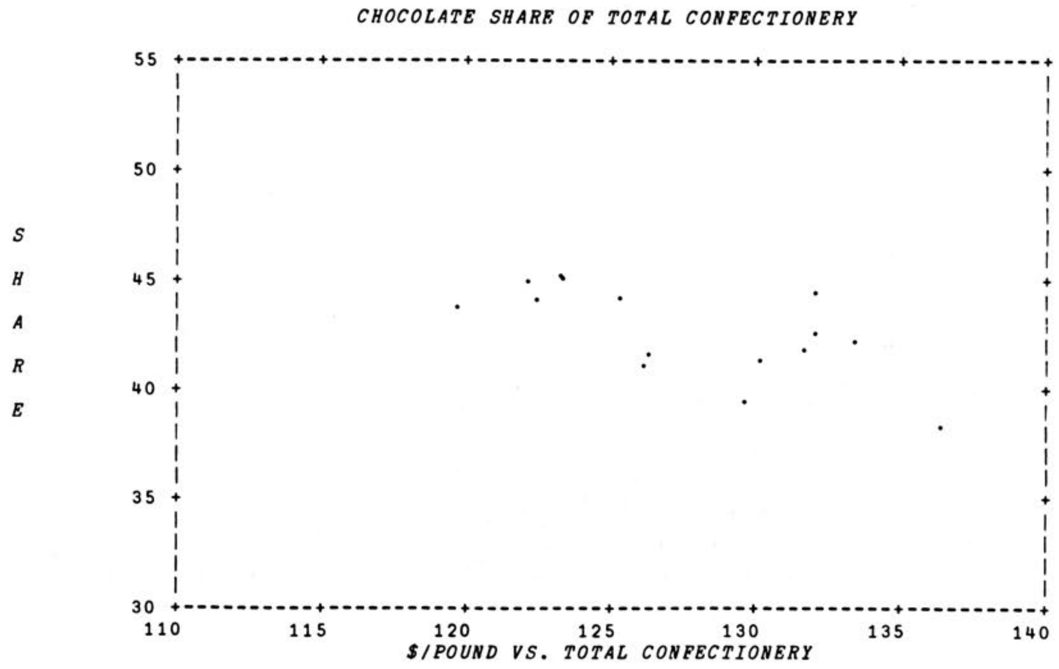


Exhibit V

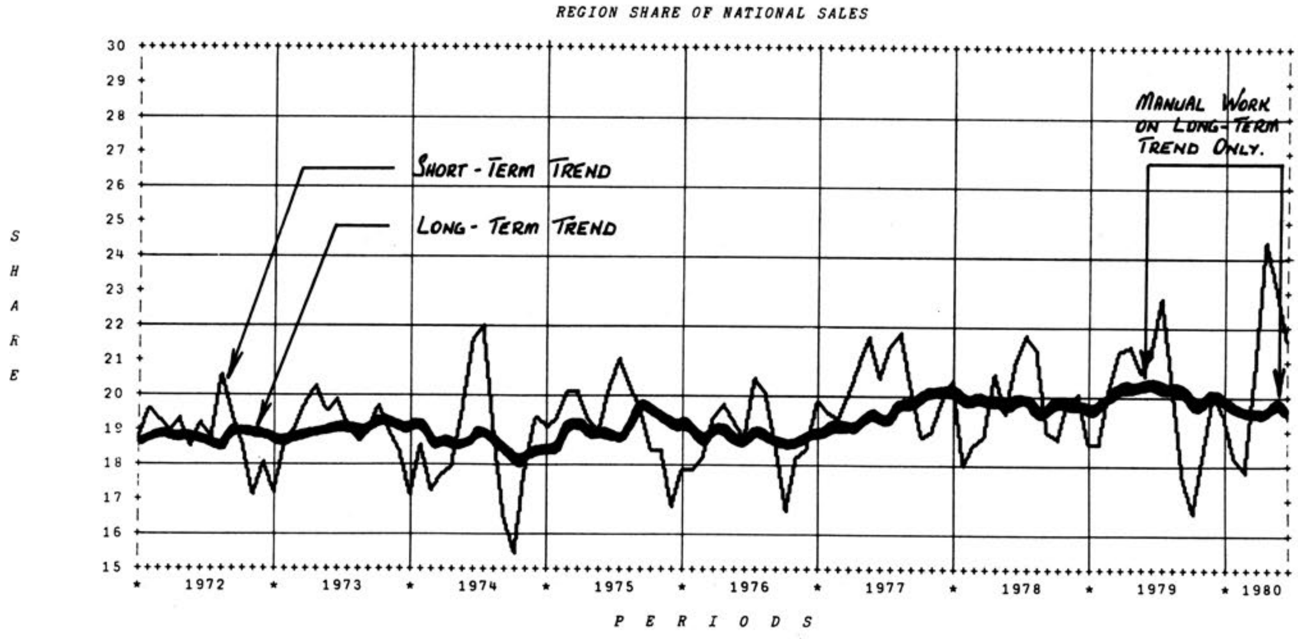


Exhibit VI

DATA MANIPULATION WITH MAGIC

David A. Keith
I.P. Sharp Associates Limited
Toronto, Ontario

Several years ago, when I was accumulating ideas for a system that has since become known as MAGIC, I wondered, "Can a system as simple as MAGIC actually succeed in the marketplace?" "Surely not", I said to myself, and in fact people might even laugh at me for proposing such a system, or more likely, for calling it MAGIC. Since then, a few people have laughed but a whole lot more have not laughed and have become serious users of MAGIC.

This fact has taught me an invaluable lesson — that we, as computer professionals, must not underestimate the importance of designing computer systems that are not only flexible, but also have appeal to the masses. By "masses", I mean those people who have problems too small for their computer department, but too large, repetitive, or boring for a desk calculator. It is my belief that the majority of people with small analytical problems are just not being served by today's current level of computer software and hardware. Hopefully, MAGIC is a system which helps to correct this deficiency.

The following examples provide a sample of how MAGIC works, and the results it can produce.

Example 1

This example shows MAGIC being used to retrieve and manipulate time series data from the International Monetary Fund's IFS data base. The 1st, 2nd, 4th and 5th columns contain data retrieved from the data base, while the 3rd and 6th columns are calculations based on that data.

```
CLEAR
TIMESERIES
MONTHLY,DATED 1 79 TO 6 80
SCALE 0
1 2 4 5 PUT IFS 'CAN,USA/71,70'
3 6 PUT (ITEM 2 5) MINUS (ITEM 1 4)
TITLE 'IMPORTS AND EXPORTS'
3 3 HEADING 'CANADA (MILLIONS),U.S.A. (BILLIONS)'
LABEL 'IMPORTS,EXPORTS,BALANCE,IMPORTS,EXPORTS,BALANCE'
'HP' TABLE ABOVE
```

```
                         IMPORTS AND EXPORTS

------------------------------------------------------------------
        |     CANADA (MILLIONS)       |    U.S.A. (BILLIONS)
        |-----------------------------|---------------------------
        | IMPORTS| EXPORTS| BALANCE| IMPORTS| EXPORTS| BALANCE
------------------------------------------------------------------
  JAN/79    5,102    5,263      161   16,873   12,561    (4,312)
  FEB/79    4,776    4,781        5   14,628   12,932    (1,696)
  MAR/79    5,883    5,658    (225)   16,748   15,587    (1,161)
  APR/79    5,244    5,132    (112)   17,195   14,267    (2,928)
  MAY/79    6,037    6,076       39   17,529   14,819    (2,710)
  JUN/79    5,455    5,559      104   18,508   15,366    (3,142)
  JUL/79    5,268    5,311       43   18,182   14,732    (3,450)
  AUG/79    5,947    5,717    (230)   19,017   15,009    (4,008)
  SEP/79    5,074    5,897      823   19,155   14,940    (4,215)
  OCT/79    6,509    6,541       32   20,367   17,283    (3,084)
  NOV/79    6,181    6,277       96   19,776   17,320    (2,456)
  DEC/79    5,067    5,922      855   20,950   16,985    (3,965)
 YEAR/79   66,543   68,134    1,591  218,928  181,801   (37,127)

  JAN/80    5,787    6,148      361   21,254   16,361    (4,893)
  FEB/80    5,747    6,405      658   21,745   16,971    (4,774)
  MAR/80    6,332    6,776      444   22,168   19,685    (2,483)
  APR/80    6,821    6,455    (366)   20,730   19,147    (1,583)
  MAY/80    6,011    6,263      252   21,708   18,770    (2,938)
  JUN/80    6,032    6,796      764   21,585   18,675    (2,910)
  YTD/80   36,730   38,843    2,113  129,190  109,609   (19,581)
------------------------------------------------------------------
```

Example 2

Although MAGIC was initially designed to handle time series calculations and displays only, it now operates equally effectively in the "cross-sectional" sense under the *NOTIMESERIES* option. This example shows *NOTIMESERIES* and *AUTOLABEL* in use with the Financial Post Corporate Data Base.

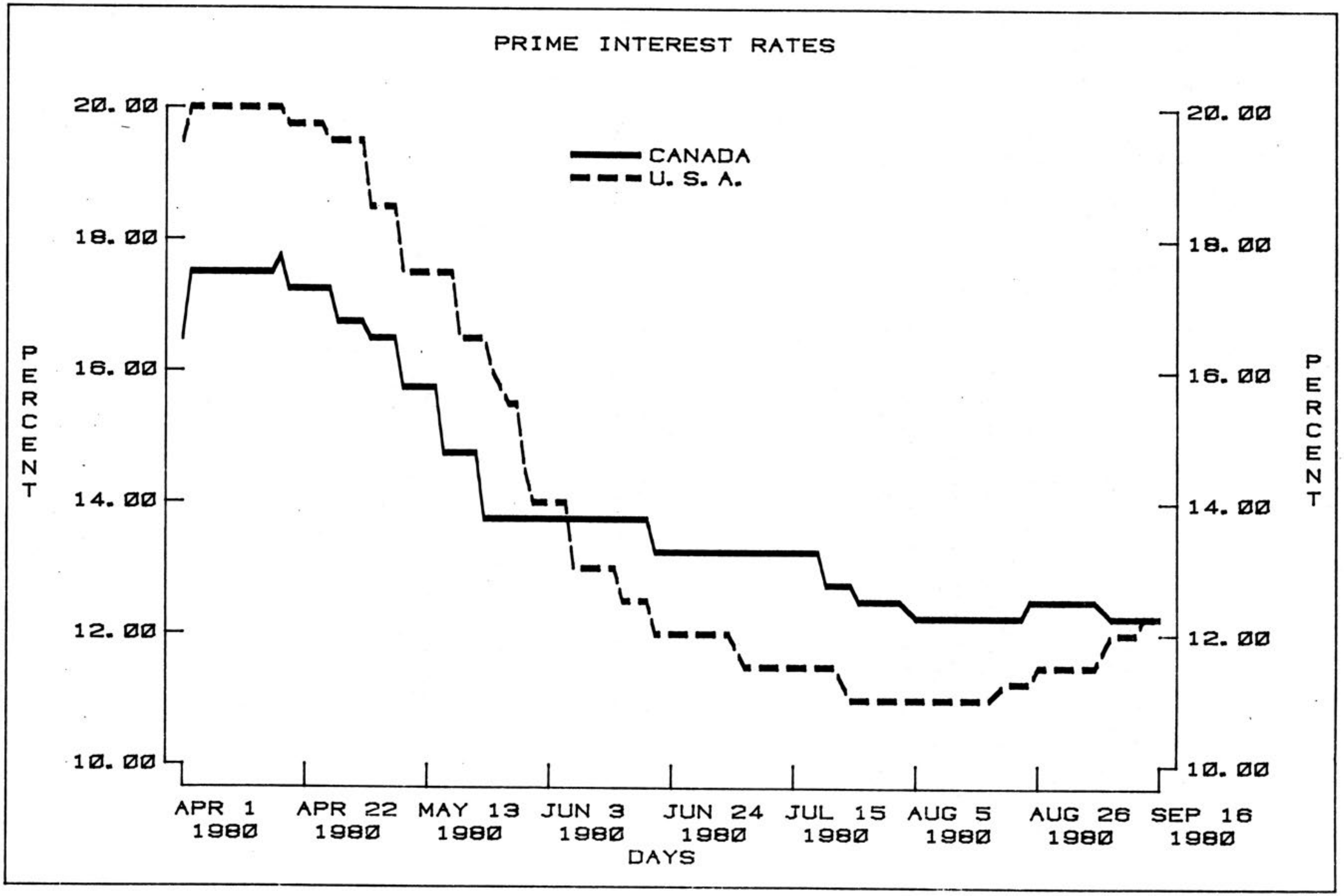
```
CLEAR
NOTIMESERIES
AUTOLABEL
YEARLY,DATED AT 79
TITLE 'INCOME STATEMENT (000) - 1979'
PUT 'BVI,DMP,NCN' FPYEARLY 29 29 31 32 33 34 52 35
                           36 37 38 39 40
2 PUT (ITEM 1) MINUS (ITEM 3)
2 COLLABEL '     NET EXPENSES'
LABELWIDTH 37
TABLE ABOVE
```

```
                         INCOME STATEMENT (000) - 1979

                                             BOW                  NORCEN
                                           VALLEY       DOME      ENERGY
                                        INDUSTRIE  PETROLEUM  RESOURCES
                                             LTD.    LIMITED    LIMITED
29. NET SALES                             275,648    945,466    887,336
    NET EXPENSES                          181,462    512,881    682,354
31. OPERATING INCOME                       94,186    432,585    204,982
32. DEPRECIATION; DEPLETION AND AMORT      30,108     62,948     44,902
33. LOAN INTEREST                          37,353    141,202     47,617
34. INVESTMENT INCOME                                 42,121
52. ADJUSTMENT FOR NON-RECURRING INCO      -3,810    -29,906
35. PRE-TAX INCOME                         30,535    300,462    112,463
36. INCOME TAXES; FLOW-THROUGH BASIS          239                15,426
37. INCOME TAXES; DEFERRED PORTION (D      13,929     95,880     24,211
38. MINORITY INTEREST                                             5,935
39. NET INCOME; FLOW-THROUGH BASIS (F      30,296    300,462     91,102
40. NET INCOME; DEFERRED CREDIT BASIS      16,367    204,582     66,891
```

Example 3

The next example shows the movement of the Canadian and U.S. prime interest rates for the past few months, using daily data from the interest rates data base (IRATE).

```
    CLEAR
    TIMESERIES
    1 2 3 4 5 DAILY,DATED 4 80
    TITLE 'PRIME INTEREST RATES'
    LABEL 'CANADA,U.S.A.'
    YLABEL 'PERCENT'
    ΔSUPERPLOT 'TERM,HP/STYLE,SOLID,SDASH/PASSES,3/SIZE,9 6'
    'LRBJH' PLOT IRATE 'CPRIH,UPRIH'
```


PRIME INTEREST RATES
CANADA
U. S. A.
20.00
18.00
16.00
14.00
12.00
10.00
PERCENT
PERCENT
APR 1 1980
APR 22 1980
MAY 13 1980
JUN 3 1980
JUN 24 1980
JUL 15 1980
AUG 5 1980
AUG 26 1980
SEP 16 1980
DAYS

Design Features of MAGIC

A) APL Consistency

I believe that a major reason MAGIC provides both flexibility and popular appeal is that it is an APL-consistent language, in the following ways:

- MAGIC statements can be entered directly at the keyboard, in "desk calculator" mode, or they can be captured in an APL function. For example, in desk calculator mode, one can say:

  ```
        DISPLAY 'IBM' FPDAILY 4
  ```

  Alternatively, one could write a small function:

  ```
      ∇ STOCK
  [1]   'ENTER STOCK CODE:'
  [2]   CODE←⍞
  [3]   DISPLAY CODE FPDAILY 4
  ```

- All MAGIC statements must adhere to APL syntax. Users can mix MAGIC and APL in the same statement:

  ```
        DISPLAY 4 GROWTH +/[2] 2 3 10⍴SALES
  ```

- MAGIC relies heavily on the concept of functions accepting arguments and returning results. In this respect, the possibility of using a function with a right-hand argument only, or with left and right-hand arguments has enhanced the scope of MAGIC:

  ```
        LABEL 'CANADA,USA,UNITED KINGDOM'
        2 LABEL 'UNITED STATES'
  ```

- MAGIC, like APL, has been designed to handle arrays of data, not just single items of data:

  ```
        DISPLAY YEAR PCHANGE ITEM 1
        DISPLAY YEAR PCHANGE (ITEM 1 2 3) MINUS (ITEM 4 5 6)
  ```

Because of these features, inexperienced users of MAGIC can sign on, load MAGIC, make some simple data retrievals, perhaps perform a few simple calculations, and display or plot the results, all in desk calculator mode. These users don't even know how to create and save a program for the next time, and in fact they probably have no interest in doing so. I estimate that perhaps half of the users of MAGIC fall into this category. More experienced users find MAGIC a quick way to achieve results with a minimum of effort. In addition, many users have found a combination of MAGIC and APL an effective method of solving much more complicated types of problems.

B) **State Setting**

There are several options or state settings available in MAGIC. An advantage of state setting in the MAGIC environment is that once the choice for a particular setting has been made, it remains set, and does not need to be respecified. Very few other languages can offer this feature, since they do not offer the concept of global variables in a workspace. Thus, whereas other systems might require a user to specify a date range for every procedure, a MAGIC date specification can be made once, at the outset of a particular retrieval. It remains set until it is necessary to change it.

A second advantage is that default values for all options are prespecified in MAGIC workspaces. This means that if a default setting for an option is suitable, the user can leave it as is. In most cases, options have been pre-set to their most commonly used values, so that few, if any, changes are necessary. An example of this is the *FISCALSTART* option. Unless otherwise specified, a year is assumed to start in January. However, *FISCALSTART* can be used when a fiscal year is not a calendar year.

C) **Generalized Data Base Retrieval**

Another major reason for MAGIC's success is that it provides a single access mechanism to approximately 25 million public time series currently stored on-line at the I.P. Sharp Data Centre. The fact that users can easily retrieve data from several data bases in the same application has been useful. Before MAGIC, a number of data bases existed on the SHARP APL system, but access to these was available only through completely independent workspaces, often with varying syntaxes and little consistency. With MAGIC, all time series data bases are available in a consistent and convenient manner.

D) **Upward Compatibility**

An interesting feature of MAGIC is that since it is a macro-language, it has never suffered from redesign or growth pains. It is essentially a set of building blocks which users put together to solve their particular problems. Historically, the first MAGIC system was created in 1974, to access time series data only. The first data base it was linked to was the U.S. Civil Aeronautics Board Form 41 data base. Although several of the features of the current MAGIC system were not part of the original design, there is not a single feature of the original that is not part of the current system. This is important, for it means that MAGIC programs written six years ago still operate with either the original or the current system in place. Although this may seem trivial, in practice, it is not an easy objective to attain.

However, MAGIC has seen considerable evolution in the last 6 years. An example of how open-ended MAGIC has been in this respect is evident in the recent addition of the *NOTIMESERIES* option. Prior to 1980, users were able to produce reports and plots only if one of the dimensions was time. While this meant that MAGIC could handle time series applicatons, users had to revert back to APL to handle applicatons which required data from time series data bases, but not in time series form. To incorporate the *NOTIMESERIES* option, eight new keywords were added to MAGIC (there are now about 150 keywords). As well, without changing their *TIMESERIES* definitions, many keywords were extended to manipulate data under *NOTIMESERIES*. Thus, with *TIMESERIES* set, a statement like:

```
    PUT SUM ITEM 5 7 10
```

means "add the three time series referenced, and *PUT* that single item away as a new time series". Under *NOTIMESERIES*, the same statement means "add up the three rows (or columns) referenced, and *PUT* that single item away as a new row (or column)". The direction over which this operation occurs is controlled by which of the *ROWWISE/COLWISE* options is set.

Extreme care has been taken to ensure that enhancements to MAGIC have in no way altered previously existing definitions. This course of action was followed as a result of the observation that casual users of any system are not receptive to change, even when it can be shown that the change is for the better. This has led to an implementation philosophy for MAGIC of improvement where possible, but only if the improvement maintains upward compatibility. Perhaps one day, MAGIC will be replaced by a different system, but for now, the current philosophy will be maintained.

An Overview of MAGIC

In this section, a brief overview of MAGIC is presented, illustrating how simple the system really is.

A) Setting a Timeframe

```
      MONTHLY,DATED 1 75 TO 12 79

      QUARTERLY,DATED 1 79 TO 4 81

      YEARLY,DATED 76 TO 79

      1 2 3 4 5 DAILY,DATED 6 80 TO 10 80
```

Generally, if data is accessed, and no timeframe has previously been selected, then a timeframe is established automatically to match the timeframe of the first series encountered. Also, if summarization of a series is required (e.g. *YEAREND* figures or to convert a series from monthly to quarterly), then a rule for this conversion can be selected. The default setting is *TOTALS*, but any one of *AVERAGES*, *DAILYAVERAGES*, *FIRSTVALUE*, *LASTVALUE*, *MINIMUM*, or *MAXIMUM* can be selected.

Normally, once a timeframe is selected it stays set for the duration of the problem being solved. However, settings like *TOTALS* or *AVERAGES* can be altered between data accesses to advantage. For example, *TOTALS* is often useful for a flow-type of series, such as barrels of crude oil production; whereas *AVERAGES* may be more useful for stock-type series, such as population figures, or consumer price indices.

B) Accessing Data

After a timeframe is set, direct access to any of the public time series stored at the I.P. Sharp Data Centre is possible. Generally, there is one access function for each of the data bases, and its name is the same as the data base name. Data base access functions take arguments and return results.

```
      YEARLY,DATED 74 TO 79
      IFS 'USA/64'
91.6  100  105.8  112.7  121.2  134.9
```

```
      IFS 'USA,U.K,SWE/64'
91.6  100  105.8  112.7  121.2  134.9
80.5  100  116.5  135    146.2  165.8
91.1  100  110.3  122.9  135.1  144.9

      ρIFS 'USA,U.K,SWE/64'
3  6
```

These examples show how close MAGIC is to APL. The results of a data access are immediately available. They may be stored as an APL variable, or passed on to another MAGIC keyword. In this next example, the result of the previous access is passed onto MAGIC's *DISPLAY* keyword.

```
      DISPLAY IFS 'USA,U.K,SWE/64'

               1974    1975    1976    1977    1978    1979
LINE 1        91.60  100.00  105.80  112.70  121.20  134.90
LINE 2        80.50  100.00  116.50  135.00  146.20  165.80
LINE 3        91.10  100.00  110.30  122.90  135.10  144.90
```

C) **Titles and Labels**

In order to clarify the data being displayed in the previous example, it is possible to specify titles and labels, as follows:

```
      TITLE 'CONSUMER PRICES (1975=100)'
      LABEL 'U.S.A.,U.K.,SWEDEN'
      DISPLAY IFS 'USA,U.K,SWE/64'

                CONSUMER PRICES (1975=100)

          1974    1975    1976    1977    1978    1979
U.S.A.   91.60  100.00  105.80  112.70  121.20  134.90
U.K.     80.50  100.00  116.50  135.00  146.20  165.80
SWEDEN   91.10  100.00  110.30  122.90  135.10  144.90
```

Alternatively, the output may be shown with time going down the page, instead of across, using the *TABLE* keyword:

```
      TABLE IFS 'USA,U.K,SWE/64'

 CONSUMER PRICES (1975=100)

        U.S.A.      U.K.    SWEDEN
1974     91.60     80.50     91.10
1975    100.00    100.00    100.00
1976    105.80    116.50    110.30
1977    112.70    135.00    122.90
1978    121.20    146.20    135.10
1979    134.90    165.80    144.90
```

D) **Plotting**

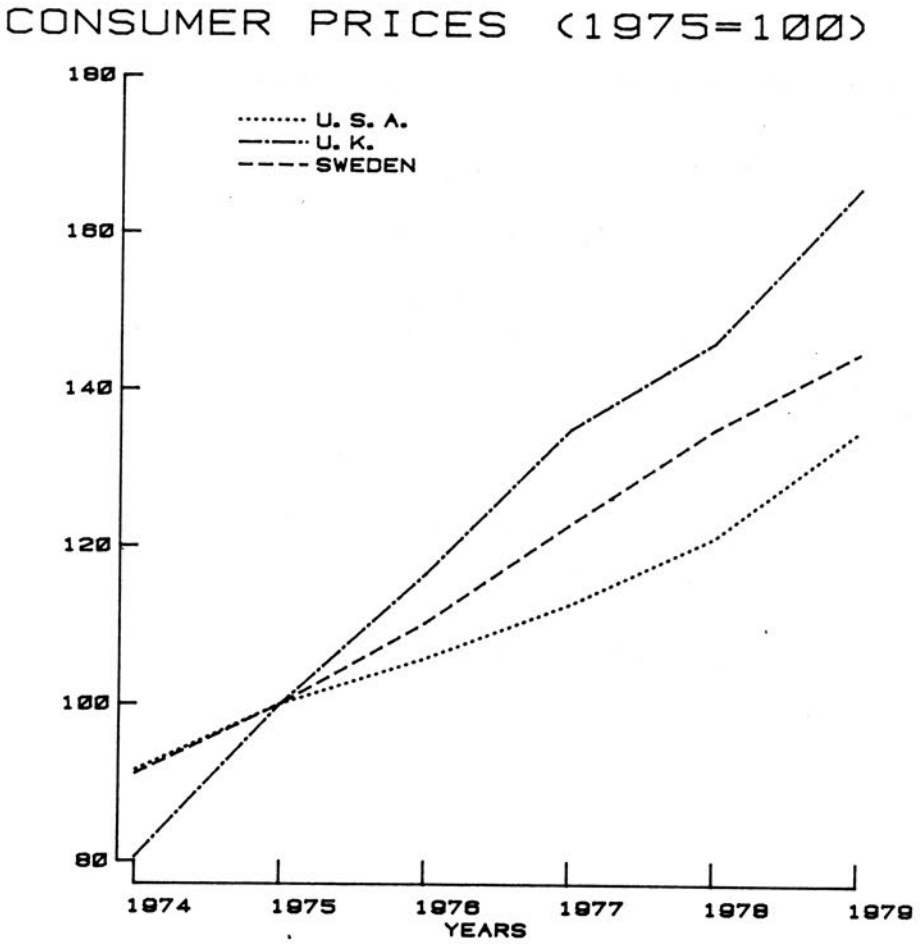
MAGIC provides a simple plot system which allows for plots on any terminal device, even the old TTY33's.

```
PLOT IFS 'USA,U.K,SWE/64'
```

```
               CONSUMER PRICES (1975=100)

                       YEARS→
   1974    1975    1976    1977    1978    1979
    +-------+-------+-------+-------+-------+-------+-------
200+
   I
   I
   I
   I
180+
   I        U.S.A.:°
   I        U.K.  :*
   I        SWEDEN:O
   I                                        *
160+
   I
   I
   I                                *
   I                                        O
140+
   I                        *       O       °
   I
   I
   I                        O
120+                                °
   I                *
   I                O       °
   I
   I                °
100+        ●
   I
   I°
   I
   I
 80+*
    +-------+-------+-------+-------+-------+-------+-------
   1974    1975    1976    1977    1978    1979
                       YEARS→
```

However, a more sophisticated package for plotting is also available with MAGIC, known as *SUPERPLOT*.

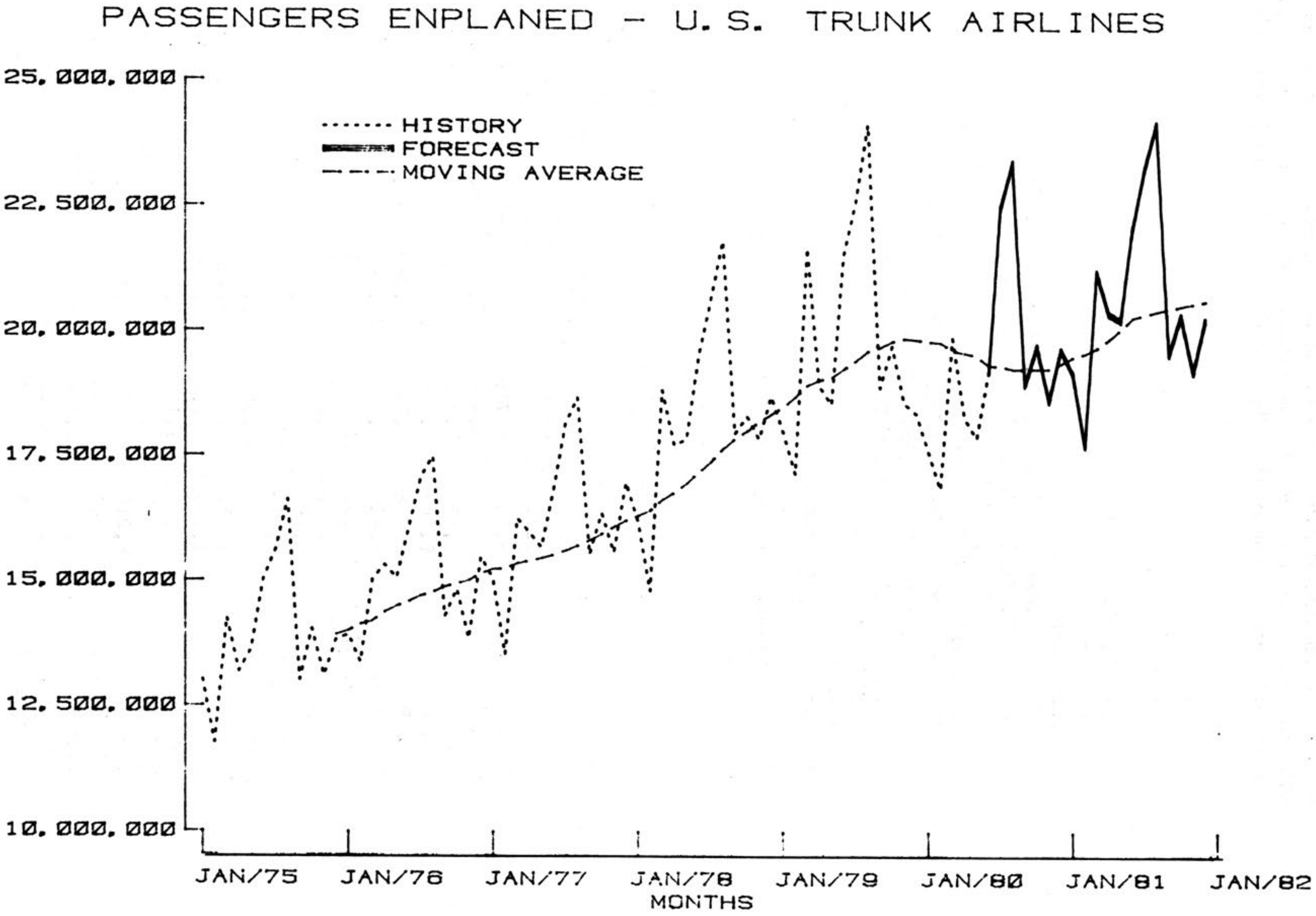
```
ΔSUPERPLOT 'TERM,HP/FONT,TITLE,2/SIZE,6.5 7'
PLOT IFS 'USA,U.K,SWE/64'
```



E) **Manipulating Data**

A distinct advantage of MAGIC is the fact that data retrieved from a data base may be easily manipulated. The next example shows the retrieval of a seasonal series which is the number of passengers carried by month by the U.S. trunk airlines. The timeframe is then extended into the future, and MAGIC's *PREDICT* function is applied to the history to obtain a seasonal forecast for the future. A 12-point moving average is then applied to both the historical and forecast data.

```
CLEAR
TIMESERIES ◇ NOYEAREND
MONTHLY,DATED 1 75 TO 6 80
PUT T1,TKS,Z110
EXTEND TO 12 81
PUT 2 PREDICT ITEM 1
PUT 12 MOVAVG ITEM 2
DATA[3;ι65]←0
ΔSUPERPLOT 'TERM,HP/PASSES,1 3 1/STYLE,DOT,SOLID,SDASH'
TITLE 'PASSENGERS ENPLANED - U.S. TRUNK AIRLINES'
LABEL 'HISTORY,FORECAST,MOVING AVERAGE'
PLOT ABOVE
```


PASSENGERS ENPLANED - U.S. TRUNK AIRLINES
25,000,000
22,500,000
20,000,000
17,500,000
15,000,000
12,500,000
10,000,000
HISTORY
FORECAST
MOVING AVERAGE
JAN/75
JAN/76
JAN/77
JAN/78
JAN/79
JAN/80
JAN/81
JAN/82
MONTHS

It is also possible with MAGIC to perform simple arithmetic calculations to calculate differences, percentages, ratios, etc. This next example shows calculations based on data retrieved from Sharp's Energy Data Base (in this case, the OECD Quarterly Oil Statistics).

```
CLEAR
NOTIMESERIES
COLWISE
YEARLY,DATED AT 78
AUTOLABEL
LABELFORM FULL
CC←CAN,US,JAP,AUL,NZ,AUS,BEL,DEN,FIN,FRA,GER
CC←CC,GRE,ITA,NTH,NOR,POR,SPA,SWE,SWI,UK
PUT QOS,TOT,CC,EXP,IMP,ROUT
PUT (ITEM 2) MINUS (ITEM 1)
PUT 100 TIMES (ITEM 4) DIVIDED BY (ITEM 3)
COLLABEL 'NET IMPORTS,NET IMPORTS AS ∘/∘ OUTPUT'
TITLE 'REFINED OIL PRODUCTS - OECD COUNTRIES'
3 TITLE 'YEAR 1978'
ROWWISE
'H' DISPLAY 5 SORT ABOVE
```

```
        REFINED OIL PRODUCTS - OECD COUNTRIES

                       YEAR 1978

-----------------------------------------------------------------
                |         |         |          |         |     NET
                |         |         |          |         | IMPORTS
                |         |         | REFINERY |     NET |  AS ∘/∘
                | EXPORTS | IMPORTS |   OUTPUT | IMPORTS |  OUTPUT
-----------------------------------------------------------------
 NETHERLANDS       29,505     9,716     58,202   -19,789    -34.00
 BELGIUM           15,027     8,476     32,396    -6,551    -20.22
 ITALY             22,773     7,548    113,955   -15,225    -13.36
 CANADA             6,148     2,799     88,430    -3,349     -3.79
 FRANCE            12,856     9,294    117,269    -3,562     -3.04
 AUSTRALIA          3,860     3,058     26,484      -802     -3.03
 UNITED KINGDOM    13,190    11,585     95,478    -1,605     -1.68
 SPAIN              2,099     2,272     49,192       173      0.35
 GREECE             3,453     3,824     11,447       371      3.24
 UNITED STATES     10,690    76,356    775,899    65,666      8.46
 JAPAN                627    23,946    232,260    23,319     10.04
 NORWAY             1,952     2,844      8,412       892     10.60
 FINLAND            1,733     3,028     11,104     1,295     11.66
 PORTUGAL             166     1,173      5,954     1,007     16.91
 AUSTRIA              177     2,388     10,032     2,211     22.04
 GERMANY            6,216    42,000    106,188    35,784     33.70
 NEW ZEALAND           41     1,160      2,955     1,119     37.87
 SWEDEN             2,865    12,574     15,330     9,709     63.33
 DENMARK            1,975    10,708      8,022     8,733    108.86
 SWITZERLAND           51     9,471      4,236     9,420    222.38
-----------------------------------------------------------------
```

F) **Private Data Files**

MAGIC processes private data as effectively as public data. In fact, a combination of private data mixed with data from the public data bases has often proven useful. Within MAGIC, the GENIE system allows for the conversational maintenance of private data bases. Several users of MAGIC (especially those who lease MAGIC for use on their in-house SHARP APL system) use it solely to analyze and plot their own private data.

Summary

MAGIC has provided a simple introduction to computers and data bases to hundreds of people over the past several years. The fact that MAGIC evolved in an APL environment during those years is in no small way a reason for its acceptance. Many users have commented that they like MAGIC because it is **simple**. The balance between simplicity and flexibility is crucial to the success of a general package such as MAGIC.

THE EUROPEAN REFINERS' MARGIN

J.A. Nasmyth
Europ-Oil Prices
London, England

Europ-Oil Prices is a London-based organisation specialising in market information for the Western European oil market. They produce two journals: **Petroleum Argus** and **Europ-Oil Prices**. The European refined product prices, as reported by **Petroleum Argus**, are also summarised in the American journal: **Petroleum Intelligence Weekly**.

****

Western Europe has 20.5 million barrels a day of installed distillation capacity — about one quarter of the world total. This vast investment in plants and in sites has become the victim of historical change. When the growth in European consumption of petroleum was checked in 1974, the capacity became excessive. In most years since then the distillation capacity has operated at break-even point, that is, with PROCEEDS from sales that have covered current costs but made no return on capital. There are only two ways in which refining in Europe can be made profitable: the first is by restrictions on crude runs and the second is from a shortage of crude. In 1978, both these things happened with dramatic results for European refiners' MARGINs in the first instance and for the rest of the world thereafter.

Looking back, it is possible to see that the European refiners' MARGIN has been critical for the world petroleum economy. As long as it remained small and approaching zero there was no pressure on spot crude prices. But when, after years of hard times, the European refiners began to make money in the second quarter of 1978, as a result of a refinery runs control plan promoted by the officials of the European Community in Brussels, there was immediate increased demand for spot crude. The revolution in Iran followed, cutting the supply of crude and, with that, the situation was out of control on a global scale.

When the European refiners' MARGIN is positive, there is the likelihood of an explosion in crude prices. When it is negative, crude prices are likely to fall in constant dollar terms. This paper describes the technique that has been evolved in the industry to measure the European refiners' MARGIN.

Europe is divided into many distinct national markets. Despite the efforts of the European community, these still have their own currencies and their own tax systems. A list of the prices of petroleum products in the European countries would make bewildering reading — different money, often different units, different specifications. A comparative student of these prices would soon find more difficulties. The legislation differs and so do the petroleum economies. In some countries it is a market economy; in others it is state-controlled.

Happily, there is a way out of this analytic difficulty and quite a simple one. While each national European market is different, they are all part of a common petroleum economy and what is happening in the common petroleum economy can be analysed by a study of the international spot market. What happens today on the international spot market will be reflected in a matter of months in the national European markets. In some, where the national market is competitive, as in Germany and Britain, the reaction is swift. In others, such as France and Italy, where the prices are government controlled, the reaction is delayed, but it comes nevertheless. All these fish are swimming in the same medium and they cannot isolate themselves from what goes on in it.

Petroleum Argus, a daily market report, is a tool for such analysis. It reports on spot prices for selected refined oil products in Northwest Europe and the West Mediterranean, together with a summary of crude oil trading. This data is available through workspace 121 ARGUS on the I.P. Sharp time-sharing system as a 2-3 page report which includes a table of oil product spot prices. This data is complemented by **Europ-Oil Prices** which comes out once a week, with a mid-week supplement, and is mailed worldwide.

This rapid communication of "hot data" about the current state of the oil market is an invaluable service. Old information, like yesterday's newspaper, is of considerably less value.

The international spot market deals in two units: in cargo loads at the main seaport refining centres, and in RHINE barges. The market in RHINE barges tends to get most of the publicity due to its concentration at the mouth of the RHINE at the port of ROTTERDAM, but it is a local market and I will talk first about the more general cargo market and return later to the barges.

Cargoes may be quoted FOB or CIF any Western European port having either refineries or oil storage, but tend to divide into two main centres, NORTWEST EUROPE and the WESTERN MEDITERRANEAN. The consumption is larger in NORTHWEST EUROPE but it is the WESTERN MEDITERRANEAN that has more surplus refining capacity and which tends to be the supplier of the last resort to the Northwest. On the principle that if you want to know what is happening it is better to get into an upstream posture, I will concentrate on the refineries of the WESTERN MEDITERRANEAN. The most important of these are on the Italian islands of Sicily and Sardinia. The refineries were built to supply the export trade — NORTWEST EUROPE primarily but also Africa, South America, the Atlantic Coast of the U.S.A. and even on some occasions, Japan.

Time series data bases on the Sharp system track the following prices:

* ARGUS - daily spot prices for selected refined oil products in NW EUROPE and WEST MEDITERRANEAN (Italy)
  for: FOB/CIF, BID/ASK, GAS OIL/MOTOR GASOLINE/ NAPHTHA/JET FUEL/HEAVY FUEL OIL (high and low sulphur). See REPORT 1.

* RHINE - twice weekly RHINE barge spot prices at ROTTERDAM, The RUHR and BASLE

for: GASOIL and various grades of MOTOR GASOLINE

These prices may be taken as representative of what prices will be throughout Europe some months later, allowing for local distortions. Refiners' revenue depends both on products prices and on the proportions of the different products he can get out of a barrel of crude. What is interesting to me is the lowest cost supply and that is the production that can be obtained from a distillation column with a minimum of product rectification. I therefore assume such a distillation product yield and for crude, I take ARAB LIGHT, which is OPEC's marker crude.

A Mediterranean refiner's PROCEEDS from running a barrel of ARAB LIGHT and selling the products at spot cargo prices are calculated (Report 2) and are plotted in Graph 1. These PROCEEDS rose steeply in 1979, from around $20 a barrel (or about $130 per tonne) at the beginning of the year, to $37 (or about $250 per tonne) at the end. This year they have fallen in two stages, the first in the early weeks of the year and the second from mid-summer onward. By the end of August they were a little under $29 a barrel.

Meanwhile, the price of crude has risen. To obtain a measure of the Mediterranean refiner's MARGIN we deduct from his PROCEEDS the cost of refining and the cost of freight at current spot rates from the loading terminals of the Arab Gulf. If this NETBACK exceeds the current price of crude, the refiner may expect a positive MARGIN from processing. The resulting NETBACK FOB Arab Gulf was, at the end of August, about $28 a barrel. This is the price that the companies that are shareholders of Aramco pay for this crude. Other buyers must pay for equivalent crude, a price around the OPEC marker price, which is now $32 a barrel. Report 3 shows an example of a NETBACK calculation for ARAB LIGHT refined in the West Mediterranean.

The use of APL to automate the long term analysis of our published data applying techniques such as NETBACK calculations, has provided a timely solution to analytic problems.

The following models have been implemented:

* PROCEEDS of a refiner in the Mediterranean running ARAB LIGHT Crude.
  Calculates the gross revenue for ARAB LIGHT crude and selling products of a 'composite barrel' at the daily spot prices. In APL....
  +/PRODUCTSPOTPRICES x REFININGFRACTIONS

* NETBACK of Mediterranean refiner to the Arab Gulf from PROCEEDS .
  Deducts from PROCEEDS, as defined above, the refining cost and the freight Persian Gulf to Mediterranean at current spot rates.

* NETBACKS to Mediterranean refiner of sales of Gas Oil at current prices in the Ruhr (West Germany) and New York.

* NETBACKS to Mediterranean refiner of sales of Motor Gasoline at current prices in the Ruhr (West Germany) and New York.

In addition, the MARGIN for refining ARAB light in the West Mediterranean area can be calculated from the difference between the NETBACK and the current spot price of ARAB LIGHT crude culled from the ARGUS daily market reports.

At the end of August the Mediterranean refiner's PROCEEDS had become aligned to the cost of the lowest priced crude available. For refiners running other Arab Gulf crudes, this would mean a loss approaching $4 for every barrel of crude run.

PROCEEDS of refiners supplying the inland markets of Europe were not as low as this, but they are moving downwards.

I started by saying that distillation capacity in Europe could not in normal times be expected to earn a profit because there is too much of it. But the owners will close it down rather than run it at a loss and this is now happening. The throughput of the European refineries was, on the latest count, down 9.4 percent from a year earlier.

The position at the end of August was that the product prices in the European spot market were on a weighted average below the cost of crude at the official OPEC price. If the calculation is done on the assumption that the refiner has conversion capacity, the loss will be less, but there will still be a loss. If the calculation is done based on the more expensive African crudes, costing some $37 a barrel FOB, the loss will be appreciably larger.

What effect do product prices in Europe have on mid-East crude prices?

From 1974-78, crude prices fell in constant dollars. Through this period, European product prices were on the floor set by the cost of incremental crude, which may be taken as Arab Light and spot freight costs from the Arab Gulf to Europe. Depressed product prices in Europe and stable crude (in constant dollars) went together.

In the second quarter of 1978 product prices in Europe began to rise. They continued to recover through the second and third quarter of 1978 and towards the end of that time an extraneous factor came into the equation - the revolution of Iran. Fear of what might happen if Iranian crude were lost to the West caused speculation in all forms of oil, with product prices leading the way.

Graph 2 shows that European product prices, as measured by the NETBACK of the Mediterranean refiner in the Arab Gulf, led the way up through the first part of 1979. Spot crude prices followed and official OPEC crude prices (i.e. government sales prices) followed a good way behind.

In the middle of 1979, a significant change occurred. Spot crude prices went higher than the Mediterranean refiners' NETBACK. Since that date, spot crude prices have stayed higher and, in recent months, very much higher with the result, as we have seen, that refining in Europe has become unprofitable.

From mid-1978 to mid-1979, product prices in Europe were responsible for drawing crude prices up. The OPEC states reacted only with a considerable delay to the market indicator of soaring spot prices for their crude, which was in turn responding to high product prices in Europe. But since mid-1979, the product prices in Europe have had a dampening effect on crude prices. We still wait to know if this will result in an actual fall.

Having traced the impact that Europe's excess of distillation capacity has had and may have on crude prices, the next step is to see what impact it makes on the neighbouring petroleum economy — that of North America.

For more than twenty years, since President Eisenhower made crude import quotas mandatory in 1959, the petroleum economy of the U.S. has been insulated from the rest of the world. From 1959 to 1973, U.S. domestic prices were above the international level, a result of keeping out cheap Middle-East crude. From 1974 onwards, they were below the international level, a result of price control on domestic crude. The present represents a stage of transition. U.S. product prices are usually below the international level, but they rise up to it at times of stringency when domestic supply is inadequate and imports are needed. As the price controls phase out, U.S. domestic product prices will move closer to European parities, or perhaps the Europeans will move closer to the North Americans. What this means in practice may be seen from Graphs 4 and 5. In these, the relationship between U.S. and European prices are plotted for two main products, MOTOR GASOLINE and GAS OIL.

Once again the market is considered from the perspective of a refiner in the West Mediterranean geared to the export trade. For him, the main market is in Northwest Europe, typified by the industrial concentration in the RUHR valley in West Germany. Starting with the ex-tank prices of these products in the RUHR and deducting the RHINE barge freight to Rotterdam and the sea voyage freight from the Italian islands to ROTTERDAM, we calculate the refiner's NETBACK from sales delivered in the RUHR.

This is compared with his NETBACK from sales delivered in New York, calculated from the price of barges in New York harbour as reported by **Oil Buyers' Guide** with the cost of the sea voyage from Italy deducted. (N.B. O.B.G. data is not yet one of the time series data bases.)

To consider first GASOLINE, it will be seen that prices in these two great consumption centres have kept pretty closely in step, as seen from the point of view of the refiner in Italy. In 1979 the NETBACKs kept close except in two quite short periods, in May and in November, when lines at the gas stations or fear of recurrence of lines at the gas stations caused a sharp rise in the New York barge price. Through 1979, the U.S. refinery industry was struggling to meet demand and the price at which incremental supply could be imported from Europe was an ever present factor in the market. The situation has changed completely in the present year. Since May, the New York GASOLINE price has fallen and the NETBACK from this market is now much below that from the RUHR.

As a result of the recession, the U.S. refinery industry no longer has any difficulty in meeting domestic demand and the possibility of imports has become remote.

In the case of GAS OIL, the correspondence between prices in the RUHR and in New York is not nearly so close as in GASOLINE. It was only for a few weeks in March and April of 1979 that the U.S. had need for imports and the prices in New York harbour rose above parity with the RUHR. The experience of the shortage of heating oil at that time was so traumatic that the U.S. refinery industry came under orders from Washington to make sure that there would be no repetition in the winter of 1979/80.

The close correspondence between GASOLINE prices in the RUHR and in New York and the lack of correspondence between GAS OIL prices in these two centres may be attributed to the anxiety of the Administration in Washington to ensure that GAS OIL stocks were adequate and the consequent difficulty that the U.S. refinery industry had in supplying sufficient GASOLINE.

For the time being, the two markets are independent for both of these products. The recession has ensured that domestic supply is sufficient. But this can only be a temporary phase. Presumably industrial recovery will again promote demand to the point at which imports will be needed of either or both of these products. Even if it does not do so and the demand is lasting, as it has been in Europe since 1973, the freeing of domestic crude prices in the U.S. will of itself bring the two markets together.

REPORT 1

```
                ARGUS EUROPEAN OIL PRODUCT PRICES
                ===== ======== === ======= ======

-------------------------------------------------------------
                  |                WEST|                    |
                  |     MEDITERRANEAN| NORTHWEST EUROPE|
                  |-------------------------------------------
                  |      FOB|      CIF|      FOB|      CIF|
                  |-------------------------------------------
                  | BID|ASK| BID|ASK| BID|ASK| BID|ASK|
-------------------------------------------------------------
FRI 29TH AUG 80
  GAS OIL           265 270   272 280   269 272   275 280
  MOGAS 98          310 315     0   0   315 320   323 328
  NAPHTHA           247 252   250 255   262 265   266 269
  JP1               320 327     0   0   320 327   320 327
  HFO 3.5 S         150 155   156 161   152 155     0   0
  HFO 1 S           174 176   176 179   174 176   184 189
-------------------------------------------------------------
```

REPORT 2

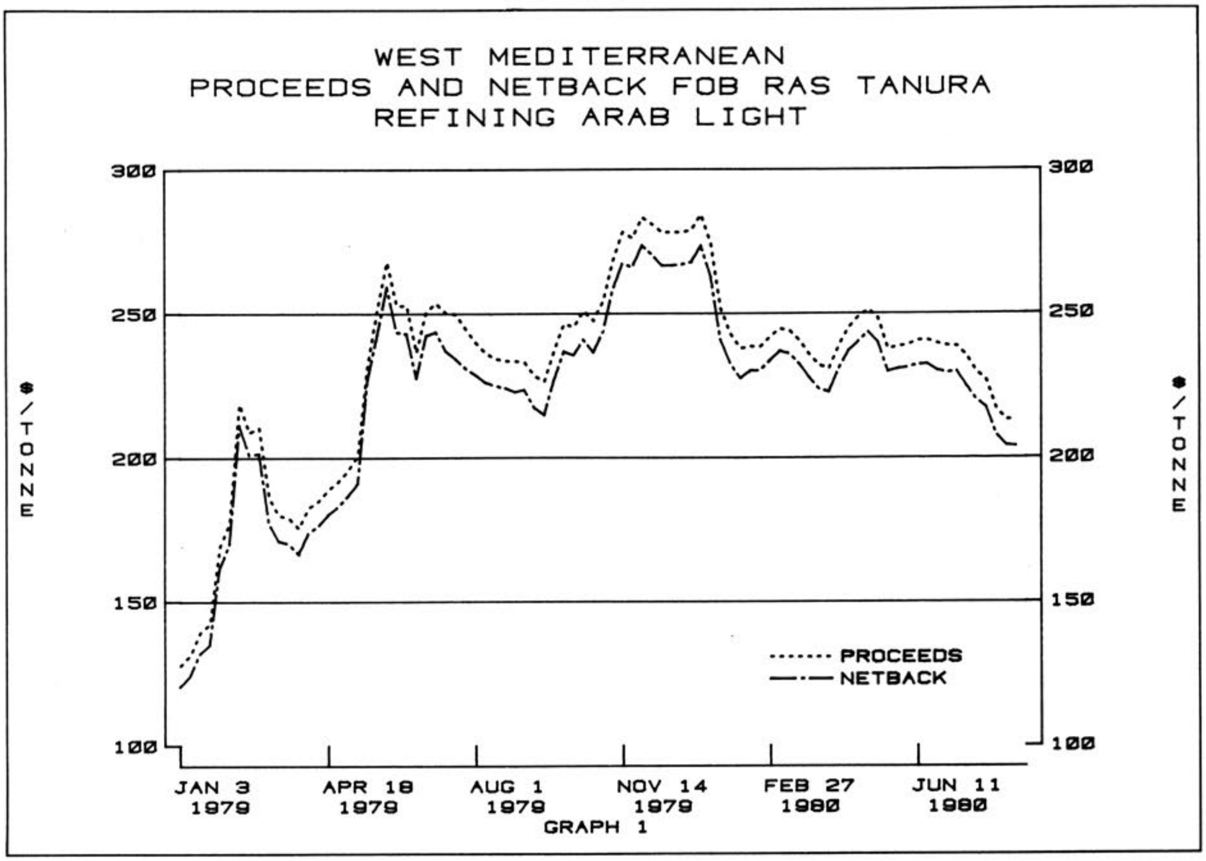
```
                          WEST MEDITERRANEAN
                          REFINER'S PROCEEDS
                            FOB RAS TANURA
                              ARAB LIGHT

---------------------------------------------------------------------
              |         |          |           |COMPOSITE|
              |         |          |           |   BARREL|
20AUG80       |      BID|       ASK|MID PRICE|      o/o|  PROCEEDS
---------------------------------------------------------------------
GAS OIL              260        265     262.50      31.50      82.69
MOGAS 98             300        310     305.00       7.50      22.88
MOGAS 92                                             2.50       7.38
LPG                                                  1.50       4.58
NAPHTHA              245        250     247.50       6.00      14.85
JP1                  325        330     327.50       4.50      14.74
HFO 3.5 S            150        155     152.50      43.00      65.58
TOTAL $/TONNE                                      100.00     212.67
TOTAL $/BL                                                     28.86
---------------------------------------------------------------------
```

REPORT 3

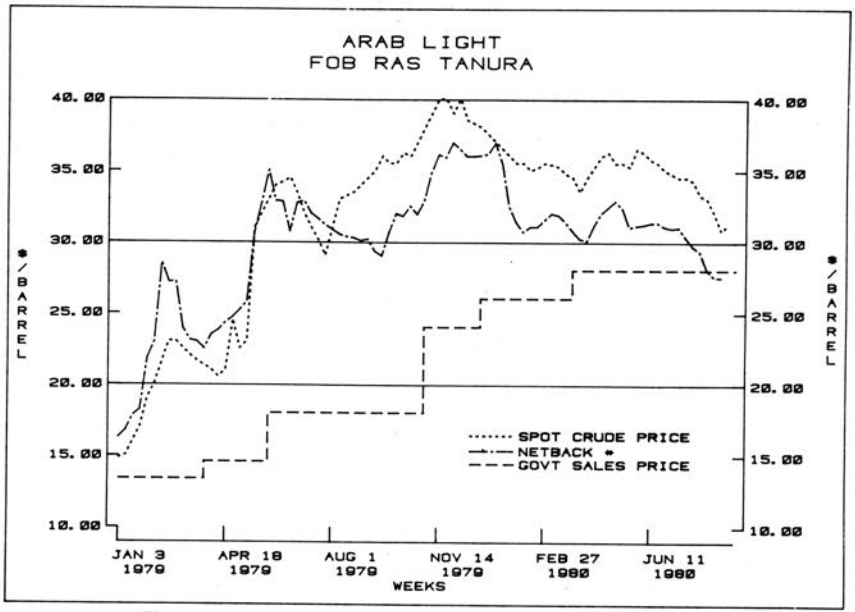
WEST MEDITERRANEAN
REFINER'S NETBACK
FOB RAS TANURA
ARAB LIGHT

	PROCEEDS	FREIGHT AND PROCESSING			NETBACK		CRUDE	MARGIN
	TOTAL	FREIGHT	REFINING COST	TOTAL	$/TONNE	$/BARREL	SPOT PRICE $/ BARREL	$/BARREL
2APR80	231.47	6.14	1.50	7.64	223.83	30.37	33.50	(3.13)
9APR80	230.26	5.95	1.50	7.45	222.81	30.23	34.50	(4.27)
16APR80	238.20	5.95	1.50	7.45	230.75	31.31	35.25	(3.94)
23APR80	244.47	5.93	1.50	7.43	237.04	32.16	36.00	(3.84)
30APR80	248.24	6.50	1.50	8.00	240.24	32.60	36.25	(3.65)
7MAY80	250.77	5.70	1.50	7.20	243.57	33.05	35.50	(2.45)
14MAY80	247.69	6.20	1.50	7.70	239.99	32.56	35.50	(2.94)
21MAY80	237.04	5.70	1.50	7.20	229.84	31.19	35.25	(4.06)
28MAY80	237.98	5.70	1.50	7.20	230.78	31.31	36.50	(5.19)
4JUN80	238.44	5.70	1.50	7.20	231.24	31.38	36.25	(4.87)
11JUN80	239.99	6.30	1.50	7.80	232.19	31.51	35.75	(4.24)
18JUN80	240.17	6.30	1.50	7.80	232.37	31.53	35.50	(3.97)
25JUN80	238.99	7.38	1.50	8.88	230.11	31.22	35.00	(3.78)
2JUL80	238.23	7.38	1.50	8.88	229.35	31.12	34.75	(3.63)
9JUL80	237.95	6.62	1.50	8.12	229.83	31.19	34.50	(3.31)
16JUL80	234.48	8.45	1.50	9.95	224.53	30.47	34.50	(4.03)
23JUL80	229.06	7.68	1.50	9.18	219.88	29.83	34.25	(4.42)
30JUL80	226.08	7.29	1.50	8.79	217.29	29.48	33.25	(3.77)
6AUG80	216.18	6.77	1.50	8.27	207.91	28.21	33.00	(4.79)
13AUG80	212.45	6.77	1.50	8.27	204.18	27.70	32.00	(4.30)
20AUG80	212.67	7.35	1.50	8.85	203.82	27.66	30.75	(3.09)
27AUG80	213.64	8.46	1.50	9.96	203.68	27.64	31.25	(3.61)

GRAPH 1

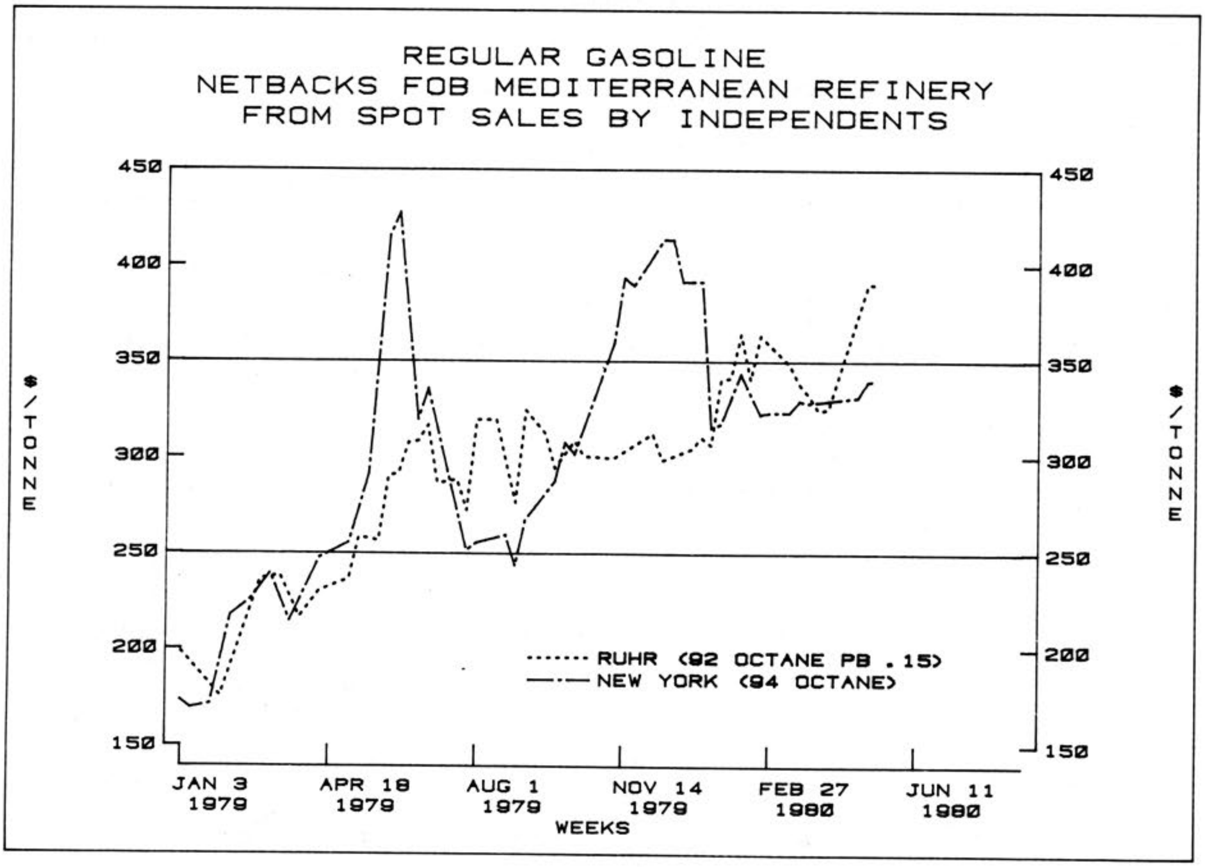


GRAPH 2

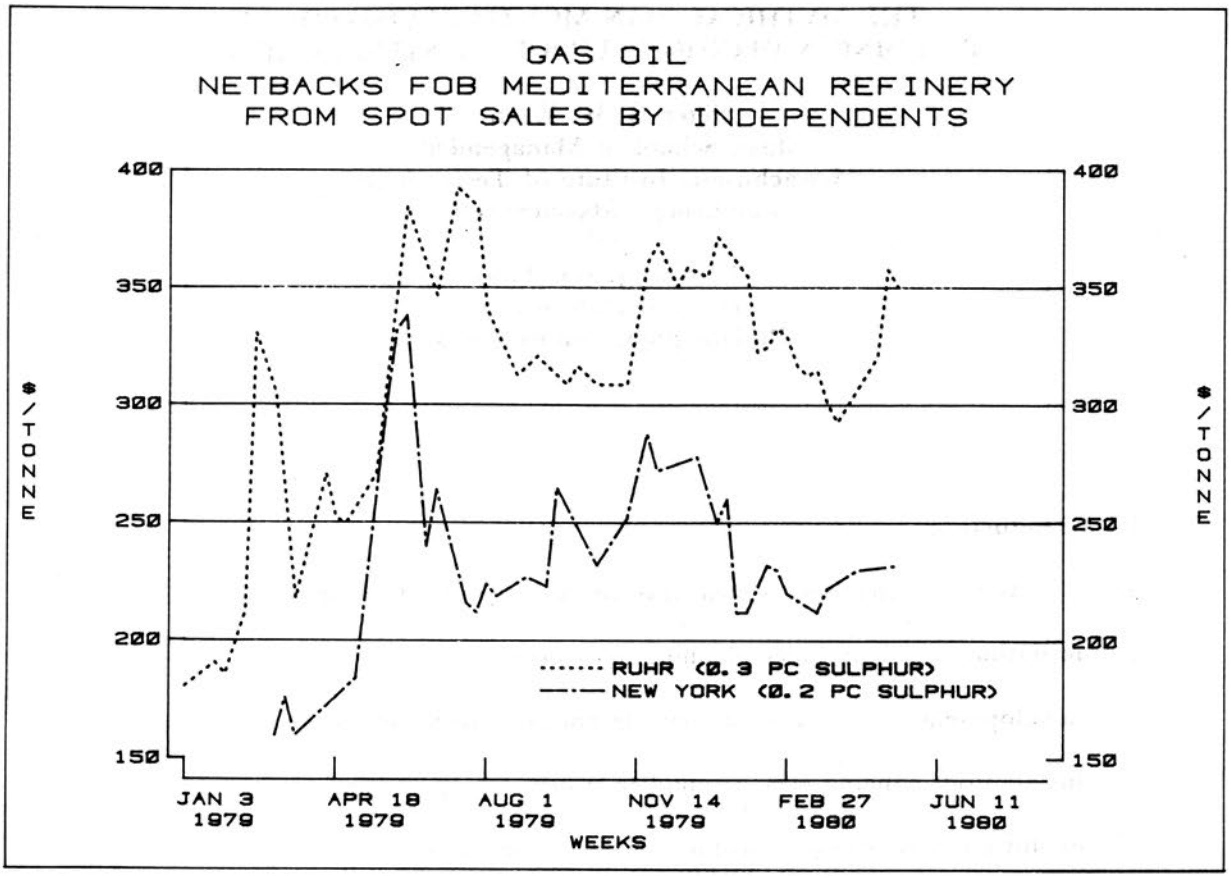


* MEDITERRANEAN REFINERS NETBACK FOB RAS TANURA

GRAPH 3



GRAPH 4


GAS OIL
NETBACKS FOB MEDITERRANEAN REFINERY
FROM SPOT SALES BY INDEPENDENTS
400
350
300
250
200
150
$/TONNE
$/TONNE
······ RUHR (Ø. 3 PC SULPHUR)
—·— NEW YORK (Ø. 2 PC SULPHUR)
JAN 3 1979
APR 18 1979
AUG 1 1979
NOV 14 1979
FEB 27 1980
JUN 11 1980
WEEKS

THE MYTHICAL MAN MONTH REVISITED: BUILDING A DECISION SUPPORT SYSTEM IN APL

Peter G. W. Keen
Sloan School of Management
Massachusetts Institute of Technology
Cambridge, Massachusetts
and
Thomas J. Gambino
The Wharton School
Philadelphia, Pennsylvania

1.1 Introduction

The life-cycle of a computer system involves a range of differentiated phases:

1) **initiation**, which includes planning and specifications

2) **development**: this covers the main technical activities of coding and testing

3) **installation**: handing over a complete system for operational use

4) **evolution**: maintenance, modification and enhancement

This systems development life-cycle (SDLC) is well understood for traditional large-scale projects. Most estimates of the relative effort needed at various stages view initiation at about 40% of the total, coding 10%, testing 30% and installation 20%. Evolution may be 100-300% of the original effort.

Interactive technology and special purpose or end-user languages are increasingly being used in applications where the life-cycle requires a somewhat different strategy. APL, for example, allows extremely fast development and modification of systems. Small-scale models can be "bread-boarded" and delivered to managers in days, with little formal planning and specification.

Decision Support Systems (DSS) are a growing example of such applications and APL is the preferred tool of many DSS builders [1]. DSS are systems for use by managers in tasks that involve judgement (See Figure 1).

DSS generally reflect an **adaptive design** [2] strategy based on:

1) a **descriptive** understanding of the way the managers think, their concepts and vocabulary, and their modes of analysis; this is needed to ensure the DSS is flexible, easy to use and control, and meshes with — supports — their existing activities

2) a **prescriptive** concept of what it means to improve their effectiveness and productivity (See Figure 2).

3) quick delivery of a **Version 0**: This is a prototype that gives managers a concrete system to react to; it provides a basic structure from which the fuller DSS can evolve through use and learning

4) relating the functions of the DSS to **user verbs**; the verbs are equivalent to statements to a staff assistant: "do this... to this"

APL is especially well-suited to adaptive design. It can be learned fairly quickly by staff analysts who are knowledgeable about the application and uses (points 1 & 2 above) and allows fast development, modification and extension (point 3). APL functions can be conveniently linked together to provide a lucid, simple dialogue (point 4).

Adaptive design implies a different balance of effort than the traditional SDLC. Instead of detailed functional specifications, the designer sketches out the user-system dialogue and identifies a limited number of funtions for Version 0. Rather than ask managers what they want, the designer in effect says, "how do you like this?" there is no "operational" system in which the user "signs-off". The SDLC is a project management technique based on getting finished — delivering the final product. Adaptive design is a strategy for getting started.

With APL, Version 0 of DSS can generally be built in under 2 weeks, assuming there are no technical or organizational problems with data management. APL is in itself many times more productive for small scale systems development than FORTRAN or COBOL [3].

There is, however, a catch in all this. The SDLC is a methodology for building a systems **product** not a **program**. Brooks summarizes the difference and its important implications for software development: "[A program] is complete in itself, ready to be run by the author on the system on which it was developed. That is the thing produced in garages...[A programming product] is a program that can be run, tested and repaired by anybody. It is usable in many operating environments, for many sets of data." [4] To become a usable product, the program must be generalizable, documented, and fully tested. A system is "a collection of interactive programs, coordinated in function and disciplined in format". The programs need a consistent syntax and semantics, and precisely defined interfaces. Brooks estimates (Figure 3) that if X is the effort to build a program, 9X is required to make it part of a systems product. This is equivalent to saying that program coding is 10% of the total time and cost, the same estimate implicit in the SDLC.

For ad hoc systems, this poses no problem, since the aim is to build only a program. DSS, however, are usually products. While fast development of Version 0 is the key to getting a usable system that can evolve, that evolution requires a stable base. Some of the DSS shown in Figure 1 have been in use in one or more organizations for up to five years.

The rest of this paper discusses the effort needed to develop a DSS product in APL. The 9X Figure Brooks cites seems to hold for APL and adaptive design as much as for the SDLC and COBOL. However, the use of APL can substantially reduce X. Brook's analysis of the "tarpit" of software engineering focuses on the Mythical Man

Month (MMM). Programmers estimate development time in man-months. These are mythical because it is the program not the product on which the projection is based.

It may well be that the advantages of APL for the early stages of software development obscure the reality of the MMM. A working system is available quickly, and if it is an ad hoc application or an experimental one, it may not be extended to the product stage.

It is important to recognize the 9X as well as the X. The SDLC took years to asticulate. Without it, costs were badly underestimated, delivery schedules unrealistic, and credibility of data-processing personnel undermined. Without some equivalent of the SDLC for full accounting and planning, even APL, with all its magic, is vulnerable to similar errors. The aim of this paper is to clarify what can be called the Adaptive Development Cycle (ADC).

1.2 ISSPA

ISSPA (Interactive Support System for Policy Analysts) is a DSS specifically built to apply and test the concepts of adaptive design [5]. ISSPA is currently in use in a number of state government agencies for school finance policy analysis. Very few of the staff have had direct involvement with computers. Their training is mainly in political science and education.

The staff's responsibilities include evaluating and "costing out" legislative proposals, providing detailed analyses of financial, economic and demographic data and responding to legislators' requests for ad hoc information. Much of their time is spent on preparing reports aimed at making a case for or against legislative changes. The quality of their work is strongly constrained by lack of reliable data, the inflexibility and poor quality of many of the computer's models and reports available to them, and the difficulty of obtaining new ones. ISSPA is designed to provide the analysts with a low cost, flexible system for information access and analysis, that requires minimum set-up, learning and technical expertise. Obviously, such a DSS is a systems-product, with all the requirements over and above program coding that Brooks identifies.

2. Initial Development

2.1 Dialogue Managers

There are three separable components of a DSS (see Figure 9)

1) the dialogue manager

2) the data manager

3) the user's operators

A DSS must be both useful and usable. Its usefulness depends on the functions for analysis, retrieval and reporting. Usability is determined by the quality of the dialogue manager. To the user, the interface **is** the system. The dialogue needs to be consistent, lucid, friendly and easy to use. Carlson's analysis of five interactive business applications [6] found that about 50% of the lines of code support the dialogue, 30% support the system operators (or "data transformation") and 20% support data management. There are two basic strategies for building the dialogue manager: APL is excellent for both:

1) menu-driven dialogue

2) command-driven

The menu approach structures the dialogue and reminds the user of the available options; he or she then **selects** the next step. In a command-driven DSS, the user **invokes** the operators and is given maximum flexibility. Each strategy has its merits [7].

For inexperienced or casual users, menus reduce the amount of learning needed and minimize errors. Experienced users will generally prefer to drive the system themselves rather than respond to a fixed sequence of questions. However, the menu approach may be better suited to a very complex application in that the overall problem may be decomposed hierarchically. Artman illustrates this point, using Sigle and Howland's "augmented" menu system, written in APL [8].

For either a menu- or a command-driven DSS, a consistent and meaningful syntax and semantics is essential. For menus, the syntax is a hierarchical presentation of available choices (Figure 5). The semantics are keywords and/or explantory phrases. Most APL command-driven DSS use a verb- noun-modifier syntax [9], for example:

verb	**noun**	**modifier**
LIST	REVENUES	
RANK	AUG SALARY	BY SIZE
DISPLAY	SALES	FOR DISTRICTS
CORRELATE	COSTS	WITH REVENUES
CORRELATE	ALL	

APL facilitates such a syntax because of the ease of linking from one APL function to another and the absence of the complex argument lists and calling sequences — used in, say, FORTRAN. The functions RANK, PERCENT and AVERAGE can be used independently or in combination:

```
RANK          SALES
RANK          AVERAGE SALES
RANK          PERCENT MARGIN BY AVERAGE SALES
```

One lesson we learned in building ISSPA is the importance of defining the syntax as formally as possible. We began with a simple verb-noun structure and added modifiers as needed. Where possible, we handled them by using a structured dialogue **within** the command:

```
CORRELATE REVENUES WITH SIZE, COST

Do you want partial correlations? (Y/N) Yes

Which variable do you wish to control for? Size
```

This posed problems later. A complex DSS is, in effect, an end user language. This user creates models and reports through a sequence of commands. Consistency in the language is essential. Many modifiers are prepositions and conjunctions:

CROSSTAB	WITH
RANK	BY
DISPLAY	FOR
SELECT	IF

These are hard to remember. The verbs are easily learned and the nouns are variables in the data base. The modifiers are somewhat arbitrary. Users are legitimately frustrated by having to recall if the command is CROSSTABS....BY or CROSSTABS....WITH. Some of the modifiers are of course noise words, but where they are not they must reflect a consistent vocabulary and command structure.

The dialogue manager recognizes verbs and links to the relevant APL functions(s). A distinctive feature of DSS is that the system evolves [10]. This means it must be easy to:

1) add new commands

2) modify existing ones

Clearly, the operators/commands must be structurally independent of the dialogue managers and data managers.

An additional reason for defining a simple, consistent syntax is to minimize the likelihood of interaction errors.

Adaptive design relies on stepwise refinement. At any stage in development, a complete, working system is available. It can be improved or modified without affecting the DSS structure. There are 3 main reasons why such modification is needed:

1) the initial version of any command reflects the designer's best guess as to what the user really needs or will accept

2) the user is fussy and quickly becomes dissatisfied with awkward routines

3) the user learns and asks for additional capabilities

Of the 48 commands currently in ISSPA, 12 were significantly modified or improved (See Figure 6). **11** were requested by the users.

2.2 Data Management

The nonroutine and evolutionary nature of DSS use pose some distinctive problems in data management. An ISSPA data base often contains eight hundred variables, which will be used in unpredictable ways and combinations. While new APL functions can be added easily, restructuring the data management routines is difficult and risky.

The technical data management routines need to support the users' view of the data. For most applications, especially in finance, statistical analysis and budgeting, users think of the data as a simple table of values: the rows are planning units (e.g., school districts) and the columns are variables (revenues, population, etc.). While the permanent data base may be large, the current working set is generally less than 20 variables; a user can interact with only a manageable subset of information [11].

The matrix of values needed for the working set of data can be generated very simply by storing the data base as a set of vectors. In ISSPA, the CHOOSE command is used to select the data items or groups of variables for the current session. The range and power of APL's matrix operators then allow data retrieval and transformations to be done in the APL workspace, obviating the need for complex data management software.

Even with this simple strategy, we found that at least two-thirds of the time involved in extending Version 0 to the current system, involved data management. The dialogue manager presents the user with a clean surface; the system "behaves" well. Errors are localized; if a new function contains a bug, there is no interaction effect. This is not true with data management routines. Errors are easily compounded and the DSS then behaves badly and incomprehensibly. With ISSPA, several of the errors were caused by multiple versions of matrices being created in transformations on or selections of data.

The need to keep track of these manipulations and to maintain a fast response time means that data management involves far more complex design and coding than any other component of the DSS.

3.3 User Commands

The dialogue and data managers provide the basic structure for the DSS, into which the user routines can be integrated. These routines are simple APL functions. Adaptive design differs from the SDLC in that it does not begin from clearly defined user-specifications. Instead, user verbs are identified. In a command-driven system the user in effect orders a staff assistant to "do this"; e.g., LIST, RANK, GRAPH, PLOT. It is hard to see how any model or reporting routine can be used unless it relates directly to a concept in the user's head. Verbs are the basic unit for analyzing user needs.

There are two types of verbs:

1) **generic**: these are used in almost any application, e.g., LIST, GRAPH, SELECT IF or PLOT

2) **application-specific**: for school finance these include statistical routines (REGRESS, CROSSTAB) and analysis of equity issues

The initial version of ISSPA contained commands (see Figure 6). A basic rule of thumb for adaptive design is **support first, extend later**. To make the DSS usable, the designer needs to identify priority verbs; over that the users either now rely on or that will provide them with clear benefits in terms of reduced effort or improved productivity. Later, as the DSS evolves, commands that introduce new concepts and techniques can be added and accepted by the users.

The school finance analysts frequently use rankings or comparisons between the top and bottom 10% of districts in evaluating the impacts of legislation. This translates into the commands RANK, TOP, BOTTOM and NTILES (4 ntiles = quartiles, 10 = deciles etc.). The provision of a few such priority commands is enough to make the DSS useful.

Generic verbs are common to most applications. We found that most commands for statistical or financial analysis or graphics are available from APL public libraries. Generally they have been properly debugged and are program products. This obviously saves development effort (see Figure 3). The user-system dialogue will often need modifying, but even so, it takes 2 - 8 hours of total effort to integrate a routine from a public library into ISSPA.

3.4 Version 0

By identifying a few key application-specific verbs and looking in public libraries for functions to support generic ones a working system can be delivered in about 2 weeks [12]. The weakest components will be the data manager, and auxiliary features of the dialogue manager, such as "help" facilities and error-trapping routines.

Version 0 is complete in itself, but is intended to change. It is only by observing the user that the designer can get a clear idea of what the important issues are, in terms of dialogue and data management and clarify what specific commands are most useful. It does not make sense to ask what users want when they have never had an interactive tool of this type and are dealing with ad hoc, situational analysis in a complex judgmental task. Phased development is essential and version 0 a starting point for learning.

Version 0 needs to be delivered quickly and cheaply. DSS often do not provide clear quantitative benefits (see Figure 2). Traditional cost-benefit analysis is inapplicable [13]. Innovations of this type are a form of R & D; managers adopt them on the basis of value, not cost or ROI. A prototype allows "trialability". It can be used to establish value. If the cost is kept low, so too is the buyer's risk. If Version 0 is not useful and usable, the venture can be written off with little financial loss or waste of time. If it succeeds, the very process of using it clarifies the costs of specifications for evolving the DSS.

Version 0 of ISSPA took 70 hours to build. It was made available to users in four states. We learned that good users are critical to adaptive design. Since a DSS does not have fixed inputs and outputs, it can be properly tested only by creative users. There are three distinct aspects of testing a DSS:

1) **debugging**: this is done by the designer and involves finding obvious errors

2) **testing logic**: while "$*36!!AW " is an obvious error, $216 instead of the correct $203 is not. Substantial knowledge of the application is needed to spot these. Many of them are generated by the users trying out new combinations of variables commands and variables

3) **testing usability**: the user has to live with the DSS. It needs to become "mundane". A mundane technology is one that one takes for granted and can use with little conscious attention. No matter how well-designed, the DSS will contain ambiguities, inconveniencies and blemishes. Only direct use can determine usability.

3. Building a Product

3.1 Consolidation

Version 0 is a program, not a product. If the basic structure is sound, it will be easy to add new commands; however, at this stage the DSS lacks user documentation and the dialogue manager is unlikely to be robust enough to cover the possible combinations of commands and data users will subject it to. We found that considerable effort was needed to consolidate the system. We fell into the trap of releasing the program as a product too early. It is so easy to repair APL code on-line that the designer easily forgets that a "bug" is misnamed; it is an **error**, not a mysterious intruder that somehow infiltrates hitherto perfect code. Version 0 builds user's confidence; even isolated errors erode it.

Building Version 0 involves directly working with the user. The designer is a facilitator and intermediary [14]. For the DSS to be a product, it has to stand by itself. The designer's knowledge and expertise have to be built into the system. This involves immense software overhead. For example, building an ISSPA data base is simple and the designer can walk through the process with the user. That is in no way the same as documenting the process or making it foolproof.

Consolidating Version 0 more than doubled the size of the program. Testing became more complex since the number of interfaces increases exponentially as new routines are added. A hierarchical structure alleviates the problem, but even so, any extension to the non-localized data management routines is likely to affect many of the commands and the dialogue manager. In testing Version 0 we found few interaction errors. Later, the flexibility of the DSS became a hindrance. Individual functions A, B, C, and D worked correctly by themselves and in the sequences A,B,C,D or A,C,B,D, but **not** A,D,C,B.

It takes very few problems to upset users. None of the ones we encountered were difficult to solve. Since many of the initial functions were taken from public libraries, the code probably contained fewer than average errors.

Some of the effort needed in the consolidation stage reflected the success of Version 0. Users, who previously had very low expectations of what a computer system could do for them, were delighted with the initial system. It is worth noting that it was the usability of the system — the quality of the dialogue managers — that encouraged them to try out the DSS and thereby uncover its usefulness. We encouraged them to criticize even small details and quickly responded to any requests for changes.

Supply creates demand. In a mundane system, minor blemishes are irritating. Our users, who had previously accepted badly-formatted batch reports, soon demanded — as they should — maximum flexibility, ease of use and aesthetic outputs.

3.2 Evolution

In addition, the users soon began to want enhancements. These fell into three categories:

1) extensions to existing commands
2) new system commands that add to usability but not to functional capability
3) personal commands, for their individual use

We also added a fourth type of enhancement:

4) commands that extend rather than support the users' existing activities

The extensions to existing commands mainly involved minor additions to functional capability. For example, the LIST command was improved by adding report totals, averages, and weighted averages. Responsive service is key to effective evolution of a DSS: if user requests are quickly met, they increasingly take an active role in extending the systems.

Many user-requested system commands are "cosmetics". They make the DSS more convenient or understandable. For example, ISSPA users requested the command CONTINUE (originally called LUNCH) to allow them to log-off and later re-enter at the point they were at, VARS & WHAT IS to obtain the full descriptors for a variable, and SYNONYM to let them define their own identifiers for variables. Such facilities seem very important indeed to users. There again, APL allows responsive service and few of the system commands take more than a few hours to build.

DSS stimulates learning. Many of the most effective uses are unanticipated and indeed could not have been predicted by the designers [15]. The rule of thumb "support first, extend later" assumes that initially, users will not be willing to adjust to unfamiliar or novel techniques for analysis. Once they gain confidence, however, they are likely to develop imaginative new approaches. 10 of the 48 ISSPA commands were requested by users and 2 of these represented new analytic techniques. Tools shape users; users need to be able to shape tools. Only with an adaptive design strategy can this dynamic feedback process be facilitated.

Version 0 builds on a descriptive understanding of the task to be supported: the main goal is usability and compatibility with the existing process. The prescriptive map focusses on changes to the process. For example, we felt that the analysis needed to address issues of forecasting, to look at broader policy issues, especially concerning the equity of various proposals for finding public education and to explore this data more imaginatively. We added EQUITY, which incorporates 10 equity measures into a simple, one-word command, and routines for Explanatory Data Analysis (EDA) [16]. Very few analysts use these new commands. They are there if they want to try them out and we expect that their ease of use and link to use verbs will stimulate experiment.

A menu-driven strategy seems less effective than a command-driven one for adding this "extended" capability. The new commands are not obtrusive and we can add any number without the user feeling threatened or overburdened. In a menu-driven system, each category of command must appear in the main menu and each individual command in submenus. Since we are beginning to add new commands for forecasting, financial analysis, exploratory data analysis and econometric analysis, the dialogue would soon become cumbersome.

About 800 hours of effort has gone into ISSPA. The program product has 48 commands; the program released for initial use, Version 0 with minor enhancements, contained 27. Brooks' point seems valid. Of the 21 new commands, only 9 really extend the usefulness of the DSS. Now that ISSPA is a stable product, we can add new commands very easily, but had we tried to do so without putting in the huge effort to turn the program into the product, we would have relived the awful history of data processing implicit in Brooks' analysis of the MMM. As it was, the success of Version 0 bullied us into a false security. Our early users had many minor complaints. They virtually all related to issues of product versus program.

3.3 Training

Users

An additional difference between a program and product concerns training. ISSPA has a simple, clear structure and almost all the commands are self-evident. We could train the initial users just by demonstrating Version 0. In several cases, after a half-hour session, they were able in turn to demonstrate ISSPA to their superiors.

At this early stage, all the users need to learn is specific commands. When the DSS is in full operational use, they have to worry about setting up data bases, cost control (hence the request for COMMAND COST & SESSION COST) and reference manuals. The system designer can invest the time "hand-holding" the initial users, and acting as a coach and interpreter. The later users have to rely on the explanations in the DSS itself and in the user manual.

Documentation is a problem for all computer products. It is expensive and generally done badly. The user manual is a critical aspect of a systems product. Unfortunately, it is hard to create for a DSS since the system is not static and lacks clear initial specifications; it is intended to change and evolve. In addition, the manual must be written from the user's perspective. Too often, manuals merely explain the software and are written by programmers who have neither the ability nor inclination to think through how to unfold the system to a nontechnical user.

There are three clear stages in learning to use a DSS [17]: learning how to:

1) **operate** it: log-on; access data; and invoke commands

2) **apply** it: understand the commands; use them to handle typical reports and analyses; interpret outputs

3) **integrate** it into one's own job; get a sense of how the commands can combine; develop personal strategies for using the DSS.

The user manual and any training sessions often address only the first stage. Given the nonroutine nature of the tasks DSS support, this does not get the user very far. With, say, an order entry system, the user need only know how to operate it: the task is highly structured and little judgement needed. Stage 2 requires experiment and often, coaching. Stage 3 relies on the user having some degree of self-confidence and expertise.

Without a good user manual it can be hard to get from stage 1 to 3. We went through three iterations:

1) the initial manual **explained** the DSS; it was adequate for stage 1, but incomplete and lacking in examples

2) we hired a professional programmer to develop a new one. It had to be scrapped, since it explained the DSS in terms of software, not usage. It began, for example, by listing the minutiae of system operation, error handling and data management. The commands, which are the users' priority were introduced later and in less detail. The manual was intimidating.

3) we finally committed the 12 weeks of effort needed to write a satisfactory manual.

This begins with a sample session and provides information for each command on:

a) format: e.g. RANK variable list [By variable]
[(A)SCENDING, (D)ESCENDING]

b) description of the output and method

c) examples of outputs

d) examples of all variants: e.g.
RANK ALL BY V101
RANK REVENUES, COSTS, DEFICIT BY SIZE

e) comments: e.g. points to note, references, formulae, caveats

The manual is the user's window into the system. It is also regarded, rightly, as the first issue for delivery when contracting to buy a DSS, and not as an add-on. We underestimated both the importance and cost of the manual. It takes at least 8 weeks of concerted effort to produce — this is 4 times that for Version 0.

4. Conclusion: the adaptive development cycle

ISSPA is a success. It has been installed on several different computers and the users' reaction has been very favorable. The issues discussed in this paper are not problems, but facts of life. With APL as with COBOL, 9X is needed to make a program into a product. There are always hidden costs. In general, APL, by its flexibility and power, reduces them but it does not eliminate the need to follow a clear sequence of development. Skipping over the Consolidation stage or leaving the manual to do later saves time now but brings trouble in the end.

The Systems Development Life Cycle is a check list to remind data processing personnel of this irritating reality. The Adaptive Development Cycle (ADC) is similarly a check list for building a DSS product. It is shown in Figure 7. No further comment is needed here, except to stress

1) where the output of a development effort is intended to be a system used over a long period of time by more than one user in more than one location, building a program, however good, is not enough

2) time estimates, budgets and implementation plans must be based on full accounting; APL is so effective in the initial stages of DSS development, the later costs may easily be overlooked.

References

1. See examples in Keen, Alter, and Bennett (ed.) and "APL and Decision Support Systems" (EDP Analyzer)

2. Apadtive design is described in detail in Keen, P.G.W., "DSS: A Research Perspective".

3. See Carlson, The APL Phenomenon in IBM, for figures. One IBM study Carlson quotes, found APL users write 4 times as many lines of code per day as COBOL programmers. Each line if equivalent to 6-10 lines of COBOL. Development time for APL is one-half to one-fifth that with COBOL.

4. Quotes below are from Brooks, F.P., The Mythical Man-Month, Addison-Wesley, Reading, Ma., 1975.

5. See Keen, P.G.W. and Gambino, T., for a fuller description of the systems and its development.

6. Carlson, "An Overview of Productivity Aids".

7. See Artman.

8. See Artman, op. cit.

9. See Keen, "DSS: A Research Perspective". Keen and Gambino provide a more detailed discussion of user verbs in relation to ISSPA; Berry and Contreras & Skertchly, all provide useful discussions of APL and verbs.

10. See Keen, op. cit.

11. See Carlson and Sutton, and Keen & Scott Morton for examples.

12. Grajew & Tolovi provide an excellent and very detailed discussion of the costs of building a Version 0.

13. See Keen, DSS and Value Analsis.

14. See Bennett, Keen: "Systems for Top Managers, a Modest Proposal".

15. Keen, "DSS: A Research Perspective".

16. See Tukey, Explorary Data Analysis.

17. See Grace and Bennett, op. cit.

Bibliography

Alter, Steven L., **Decision Support Systems: Current Practice and Continuing Challenges**, Addison-Wesley, Reading, MA, 1980.

Artman, Ira B., "Design and Implementation of a Decision Support System for Hospital Space Management", MS Thesis, M.I.T., 1980.

Bennett, J.F., editor of **Building Decision Support Systems**, Addison-Wesley Series on Decision Support.

Bennett, J., "Integrating Users and Decision Support Systems", In J.D. White (ed.), **Proceedings of the Sixth and Seventh Annual Conferences of the Society for Management Information Systems**, pp. 77-86, Ann Arbor: University of Michigan, July 1976.

Berry, P., "The Democratization of Computing", Paper presented at the Eleventh Symposium Nacional de Sistemas Computacionales, Monterrey, Mexico, March 15-18, 1977.

Brooks, F.P., **The Mythical Man-Month**, Addison-Wesley, Reading, MA, 1975.

Carlson, E.D. and J.A. Sutton, **A Case of Non-Programmer Interactive Problem-solving**, IBM Research Report RJ 1382, San Jose, CA, 1974.

Carlson, W.M., "The APL Phenomenon in IBM", edited transcript of talk given to GUIDE-41 in Denver, CO, November 12, 1975.

Contreras, Luis and R.S. Skertchly, "User Defined Simulation Languages", 1979 Winter Simulation Conference, December 1977.

EDP Analyzer, "APL and Decision Support Systems", EDP Analyzer, May 1976.

Grace, B.F., "A Case Study of Man/Computer Problem-Solving", San Jose, CA: IBM Research Report RJ 1483, 1975.

Grajew, J., and J. Tolovi, Jr., "Conception et mise en oeuvre des systems interactifs d'aide a la decision: l'approche evolutive", Ph.D. thesis, Universite de Grenoble, 1978.

Keen, P.G.W., "Decision Support Systems: A Research Perspective", CISR Paper, Sloan School of Management, 1980.

Ibid, "Decision Support Systems and Value Analysis", CISR Paper, forthcoming.

Ibid, "Computer Systems for Top Managers: A Modest Proposal", Sloan School of Management Review, vol. 18, no. 1, pp 1-17, Fall 1976.

Keen, P.G.W. and Thomas J. Gambino, **Building a Decision Support System: The Mythical Man-Month Revisited**, January 1980.

Keen, P.G.W. and M. Scott Morton, **Decision Support Systems: An Organizational Perspective**, Addison-Wesley Series on Decision Support, Reading, MA, 1978.

Sigle, J. and J. Howland, "Structured Development of Menu Driven Application Systems", APL QUOTE QUAD '79, pp. 188-195, ACM, June 1979.

Tukey, J., **Exploratory Data Analysis**, Addison-Wesley, Reading, MA, 1977.

DSS		APPLICATIONS
GADS	Geodata Analysis Display System	Geographical resource allocation and analysis; applications include sales force territories, police beat redesign, designing school boundaries
PMS	Portfolio Management System	Portfolio investment management
IRIS	Industrial Relations Information System	Ad hoc access to employee data for analysis of productivity and resource allocation
PROJECTOR		Strategic financial planning
IFPS	Interactive Financial Planning System	Financial modelling, including mergers and acquisitions, new product analysis, facilities planning and pricing analysis
ISSPA	Interactive Support System for Policy Analysts	Policy analysis in state government; simulations, reporting and ad hoc modelling
BRANDAD		Marketing planning, setting prices and budgets for advertising, sales force, promotion, etc.
IMS	Interactive Marketing System	Media analysis of large computer database; plan strategies for advertising
CAUSE	Computer-Assisted Underwriting System at Equitable	Calculate and track group insurance policy and renewals
AAIMS	An Analytical Management System	Analysis of time series data on airline industry operations (database contains airlines' reports to Civil Aeronautical Board)

Examples of DSS Applications

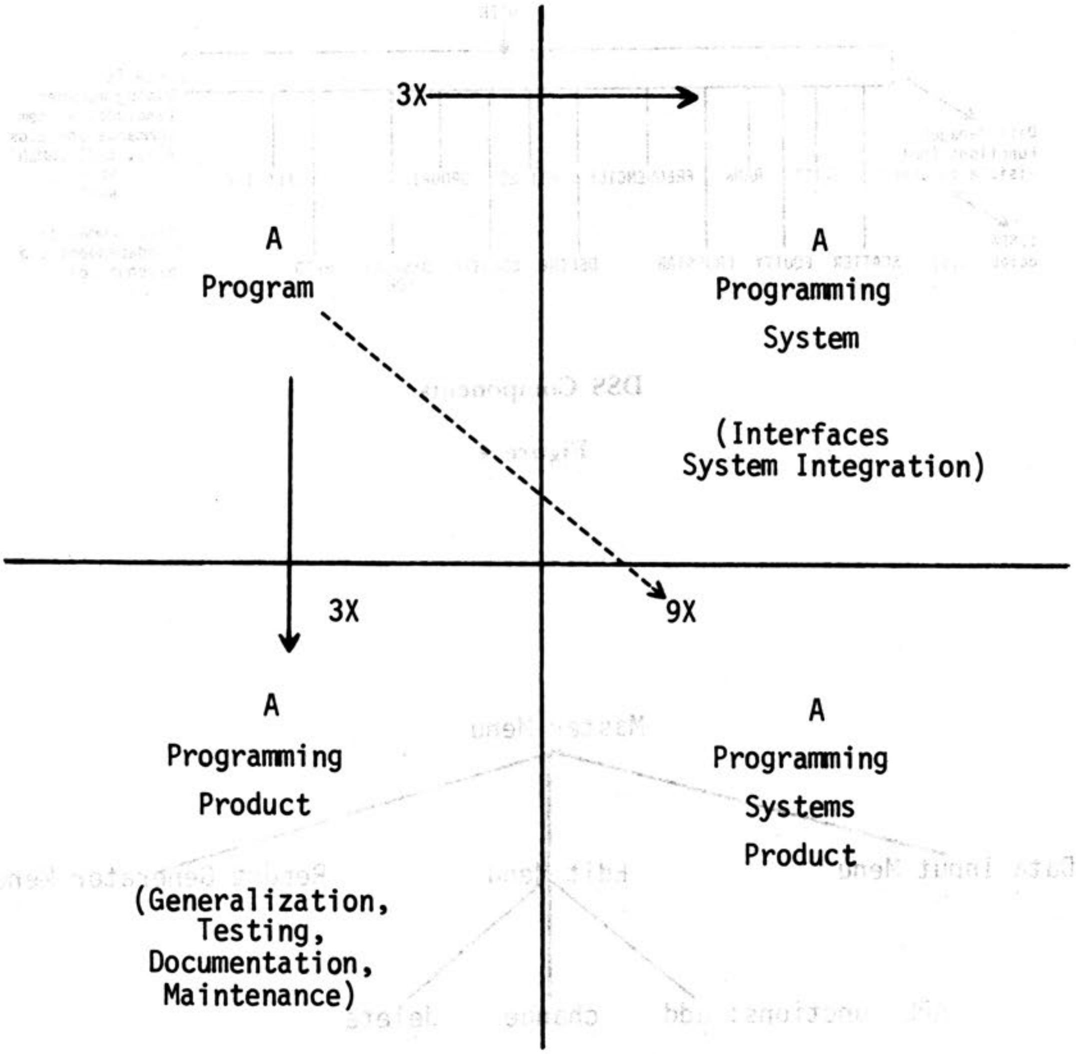
Figure 1

1. increase in number of alternatives examined
2. better understanding of the business
3. fast response to unexpected situations
4. ability to carry out ad hoc analysis
5. new insights and learning
6. improved communication
7. control
8. cost savings
9. better decisions
10. more effective teamwork
11. time savings
12. making better use of data resource

DSS Benefits

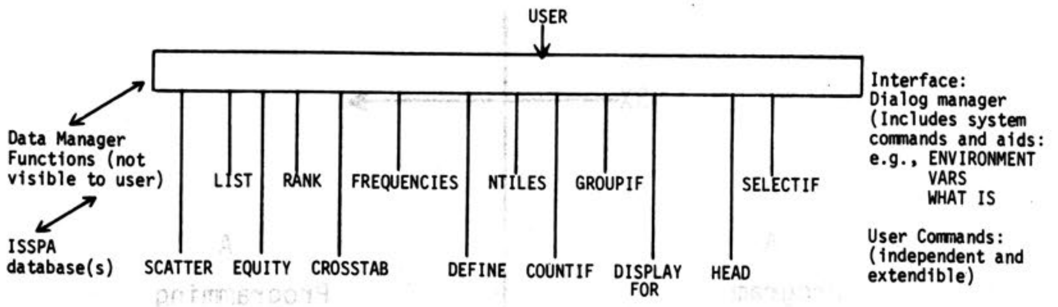
Figure 2

*** See Keen, Decision Support Systems and Managerial Productivity analysis**, pp. 31-36, for detailed examples.



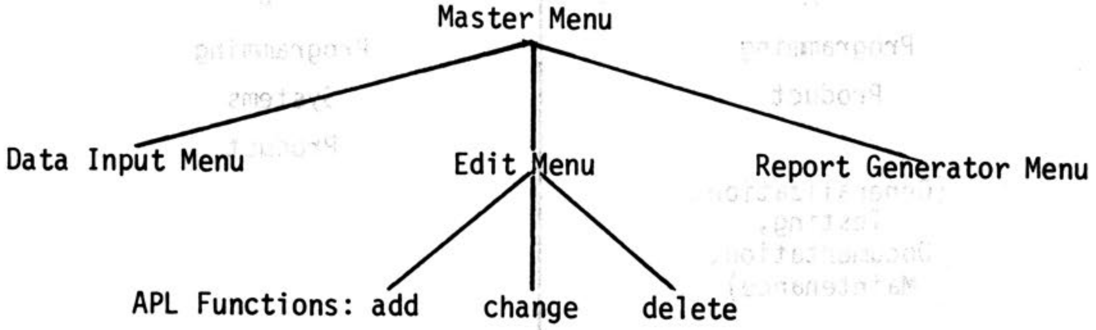
Relative Effort in Developing Systems Products

Figure 3



DSS Components

Figure 4



Hierarchical Menu

Figure 5

Examples of:	Number	Number taken from public libraries	Number later extended or modified
(1) Initial Commands			
CORRELATE, COUNTIF CROSSTAB, DEFINE, DIRECTORY, FREQUENCIES, NTILES, RANK, TOP	19	6	6
(2) Added when version made available to users			
ADD DATABASE, FORMAT SCALE, WAVERAGE	8	1	2
(3) Added at user request			
COMMAND COST, CONTINUE SYNONYM, VARS, YEARS, * SAMPLE	9	0	4
(4) Extended commands			
WTILES, EQUITY, STEMLEAF	6	0	1

ISSPA Commands

Figure 6

* Phase	
5% Initiation	- Sketch out user-system dialog - Define user view of data - Identify generic commands and priority application-specific commands - Search public libraries for APL functions - Put together "bread-boarded" version - Debug, test usability by exposure to one or more users
10% Version 0	- Release for operation by selected users - Keep in direct touch, responsing to user reactions, criticisms and requests - Add new commands for (1) usefulness (2) usability
60% Consolidation	- Freeze system - Develop full specifications for data management routines; implement and test - Add system commands ("help" features, COST commands, etc.) - Develop sample database and sample session for tutorial in manual - Develop full manual - Debug, release (system plus manual) for testing to sympathetic, knowledgeable user(s)
25% Installation	- Release for operational use - Build database(s) - Add new commands (1) user suggested (2) user-defined (3) designed-defined
Extension **	- Tailor system to expert users (e.g. "stack" commands) - Add new types of commands (e.g. for ISSPA, simulation language (IMPS) financial, econometric and forecasting routines - Improve efficiency of code and reduce operating costs

* = estimated fraction of development effort for DSS product using APL

** = openended, ongoing development

Adaptive Development Cycle

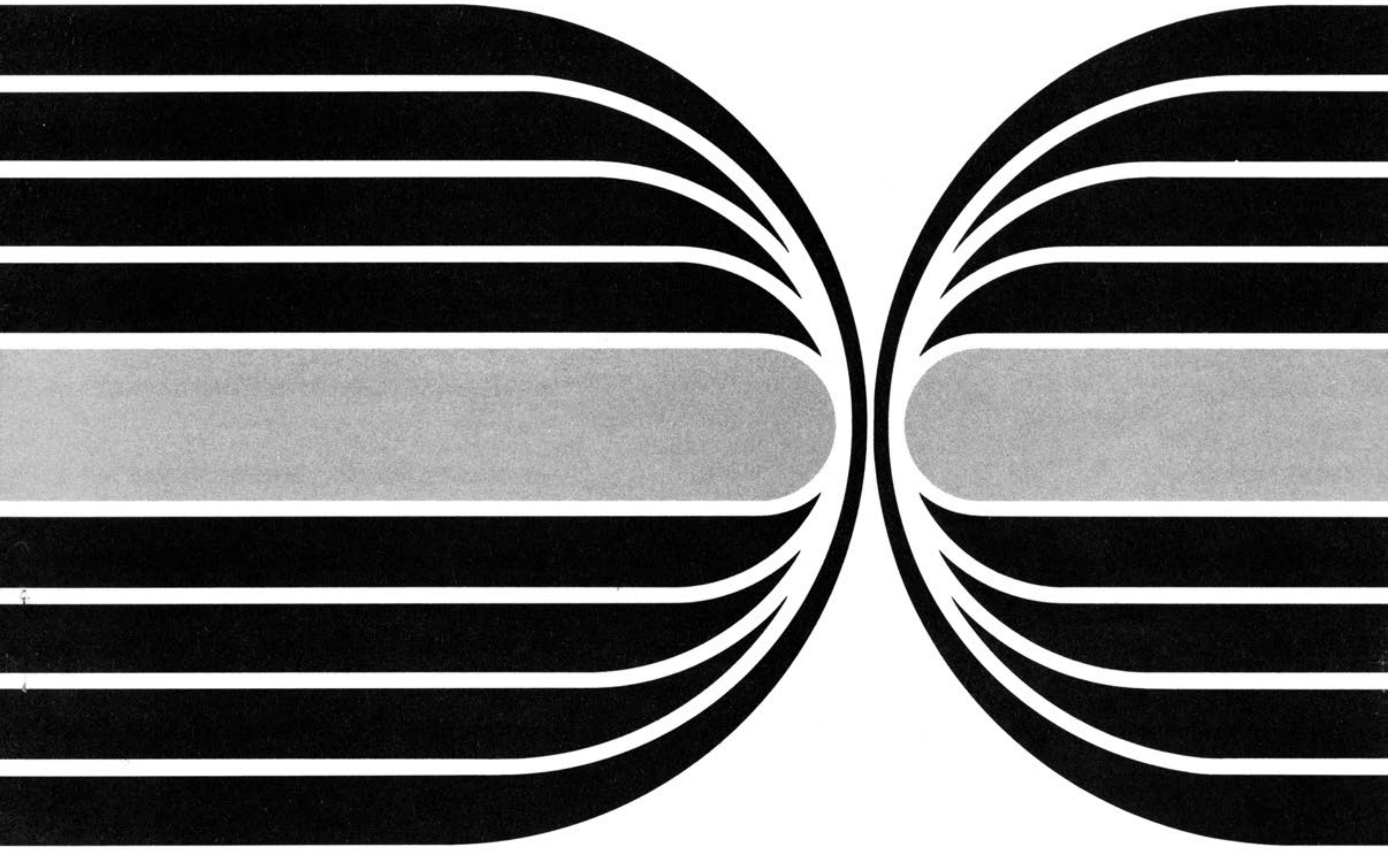
Figure 7

Printed in Canada

The following papers were presented at the **1980 APL Users Meeting** and were not available at press time for inclusion in the Proceedings.

1980 APL Users Meeting

October 6, 7 and 8
Hotel Toronto
Toronto, Canada



Sponsored by:

I. P. Sharp Associates Limited

DIMENSIONS IN ECONOMIC FORECASTING

James R. LymBurner
James R. LymBurner & Sons Limited
Toronto, Ontario

Introduction

Economists are probably one of the most misunderstood professionals there are today. So often they are perceived as having the tools of an accountant and the accuracy of a weatherman.

These misconceptions are all too much reinforced by the active use by economists of advanced computing systems and modelling techniques.

Such advanced systems have portrayed the art of economic forecasting as a natural science which is as predictable as the machinery that is used to calculate the projection. Well, we all know that is not the case. In actual fact economists are nothing more than social scientists perceiving the buying and selling behaviour of individuals in the economy.

It will be my intention in this presentation to deal with the actual classical economist's approach to information gathering, processing, and evaluating using both traditional and new computer methods. In particular I should like to deal with MAGIC or the language which appears to have bridged the gap for economists of easy retrieval, manipulation and analysis of economic pheonomena data using a dialogue which is almost non-symbolic or mathematical.

Economics and the Concept of Forecasting

Economics as it is generally perceived today is the science, both social and natural, which deals with the collective process of commerce in the economy. This in a much more concise sense, means that economics is interested in measuring and enhancing the selling and buying of goods with the smallest amount of price movement which is adverse to the general good. In this pursuit economists have a great many problems since their perception of what has happened is determined by the amount of data that can be gathered regarding the economic event itself. In most cases this event usually leaves much to be desired. The problems of data bases is in itself a whole paper and it is not my intention to expand on this topic at this time. I should only like to state however, that my perception of where the major growth and problems lie in the period ahead in economics is with data bases and their quality and speed of data accumulation.

In essence I think the message will eventually grow to be greater than the medium.

By itself, economic data is nothing more than a record of what has transpired. To become useful, economists must take the accumulated data or knowledge and use it to make forecasts of likely future events. In conceptual terms, forecasting as defined in the Oxford Dictionary is "a conjectural estimate of future things, i.e. of the coming climate". In even simpler terms, forecasting to an economist is the desire to get a quick, accurate and complete picture of the current real world's trends and project these into the future. Seems like a simple enough problem doesn't it? Well, here is where the difficulties begin for most professional economists and their credibility of forecasting starts to deteriorate. While in most cases, experienced economists are not naive enough to think that the economies are static internal relationships, the tools they have to use in their projections of the basic trends are very rigid. In other words, in most of the cases that I have examined and forecasters that I have been associated with, the rigidity of the computer model forecasting system has usually dictated a rigidity into the forecaster. The system's rigidities have most times taken the form of complexities in program design and writing. In fact, even the most experienced econometrician does not seem to adequately and completely fill this gap between system flexibility and forecasting reliability.

Much of this problem, as I perceive it, lies in the language itself. It is not in APL or COBOL or any other computer language but rather in the medium in which the economist and his economy operates, namely English, French, etc. The lack of symmetry between scientific symbolic languages and the common everyday transactional dialogue between persons in the economy and economists appears to be the point at which a lack of flexibility exists. This problem my own firm knows only too well, since we have painfully experienced it. However, when things seem the bleakest in one's search for a method often like MAGIC a solution appears.

The Economist's Application

Economist's application of computer systems to forecasting is by far the most important improvement in the art and science of economics in the last 50 years. In fact, many of our most widely accepted economic statistics today, namely Gross National Product (GNP) Account, would not be possible without the aid of computer systems. Simply the large volume of basic data on which the GNP estimates of growth, goods and services are made could not be accomplished without the aid of computer. However, while the use of computers by economists have been widespread, their success in application have only been partial. While the promises that the computer age has made to economics of making everything much more efficient has to a very large degree come true, it has yet to witness a true widespread penetration of the field. Much of the reason for these unfulfilled promises lies in the common everyday use of computers to make on-the-spot evaluations of the current economic environment. While it has been recognized that a substantial part of the problem lies with the availability of sufficient and good quality data, a major and probably "a priori" fact now is the syntax or even more basic, is a common language problem. The lack of wide and complete acceptance of computer usage in economics is due to the lack of flexibility on the part of the profession to learn a new language. While this trend is being slightly reversed by new graduates, the largest portion of economic work today is still being done only on computer system out of need, not want.

This problem, however, may be setting on the dawn of a new era in interactive languages. In this case the common vernacular of the everyday parlance of economic communication and symbol-ridden syntax of computer jargon is being bridged by a new system and non-mathematical computer prompted language.

This new system is MAGIC. MAGIC which is a by-product of APL appears to have the ability to bridge the gap between what I would call common economic dialogue and common computer language (APL). This bridging recognizes the best of both worlds while not offending either. In the case of economics the interactive communication through MAGIC is via an almost non-symbolic discussion with the system. This recognizes unknowingly the social science part of economic dialogue. This non-confrontation with a new language also encourages active use through inquiry and investigation. By keeping to standard non-symbolic interaction the economist, who is by his very nature more a social scientist, is encouraged to adapt computer usage to his everyday investigation of economic phenomena. At last, a successful marriage.

It is commonly misunderstood by the average outsider to the profession of economics that mathematics to an economist who is a social scientist is only a means to an end. MAGIC, to my way of thinking, therefore should be viewed for economists as a refinement of the means by which they can achieve their ends. The application of MAGIC is also probably true for other social sciences.

New Dimensions and the Problems

In the concluding portion of this paper I would like to dwell on the problems that economists will face and forecast under in the period ahead. Looking at this thesis in another way, one could say that what I am about to examine is the reasons why the MAGIC system in APL will probably evolve into one of the standard instruments used by economists in the future to investigate economic events.

As I envision it now, the challenges for economists in the period ahead are immense. Besides coping with our current high inflation and interest rates they will be required to analyse and forecast in a more rapidly changing environment. Extremes in the oscillations in economic variables can be expected to occur more often and rapidly in the future as monetary and fiscal policies respond to inflation more quickly.

The major problem in such an economic period will be to possess forecasting methods to adapt to such extremes. Previous methods of projecting and assessing economic growth, or the lack of it, will most likely stretch statistical methods to their limits. Wide swings, for example will force regression and other forms of statistical data fitting to become stretched in their bands of acceptable tolerance. This in turn will make the projections and forecasts based on past data more difficult. Subsequently, extrapolating from past data, possible future trends will be less and less reliable. If any one of the extremes in our economic environment are reached, namely hyperinflation, depression or deflation, it will be difficult, if not impossible, to perceive their magnitude until well after the fact.

The essence of MAGIC's attractiveness for economists in the difficult period ahead are few, but nevertheless extremely important. Foremost of MAGIC's characteristics is it's ability to help economists to respond quickly using computer assistance, to emerging trends of problem situations. Since users of MAGIC require very little dialogue translation from the event data itself to it's computer usage, the majority of the economists' time can be spent analyzing the problem rather than the computer system. As a result of this, the economist can sit down at a terminal, sign on, load GENIE if he wishes to create a file that does not already exist, load MAINTAIN to update his existing file, and then proceed to plot, table or analyse his particular information. In essence what I am saying is that the marriage that I referred to before in dialogue between the social scientist and the computer language has taken place like MAGIC. Subsequently in the future under whatever conditions should transpire, economists of any ilk will have the speed and flexibility using MAGIC to solve any foreseeable problems.

LINKING PLANNING SYSTEMS TO REAL LIFE OPERATIONS

P.H.S. Redwood
Xerox Corporation
Stamford, Connecticut

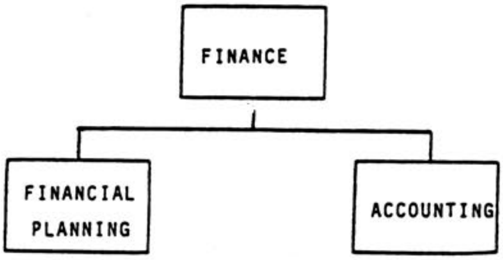
Planning vs. Accounting

"Versus" is perhaps too strong a word, but if information is properly considered a scarce resource of the corporation (i.e. there is a real cost associated with its manufacture, and the budget is not unlimited), there is a competition for this resource between those staffs who have the job of mapping the future course of the organization — the planners — and those — the accountants — who measure its performance for the benefit of management and the outside world. Of course, planners and accountants are not in open confrontation, but when their demands for information are not coordinated, a situation develops which quietly saps the information gathering resources of the corporation. As we shall see later, a second and potentially dangerous effect of competition is the emergence of alternative sources of information which produce new and different values for data items reported through conventional channels.

The reason for the apparently conflicting demands of planners and accountants is not difficult to discover. Quite simply it is that one group deals with the past — a presumably known quantity — and the other with the future, which is at best an educated guess. The two perspectives lead to fundamentally different approaches to defining, collecting, and manipulating the data each group needs to do its job. A typical result is that duplicate and overlapping mechanisms for channeling information come into being and competition develops for the services of the formal and informal people networks whose task it is to assemble and package the information upon which the two groups depend. Duplication of this kind is virtually assured when, as in most corporations, planners and accountants are part of distinct and different organizations, (even though they may ultimately report to the same department head) (Figure 1).

Duplication and overlap are patently wasteful, but there is also a more serious consequence: when two or more values for what is supposedly the same piece of information emerge from separate reporting channels, quite different conclusions may be drawn depending upon which value is recognized as "correct" by a particular group. Even if the existence of multiple values is something which is known and which requires resolution, much valuable effort may be expended in finding explanations for the apparent differences, or in determing which is "right".

It is easy to criticize this state of affairs. Indeed, in a world where in theory time is a continuous variable, the measurements of past and future are merely complementary subsets of the operations of the corporation over its lifetime. Why, then, do we have a problem? The answer is that in the real world there is an abrupt discontinuity at the point where the past and the future meet, which casts planners and accountants into quite different roles, and results in demands for different kinds of information and different ways of expressing it.



Typical Separation of Planning and Accounting

Figure 1

The discontinuity manifests itself in several ways. First and foremost there is the above mentioned need for different kinds of information. Part of this difference is perceived, and can be eliminated by education, or by some transformation, and part is real, the outcome of different professional styles and conventions. Invariably, the past is recorded in some considerable detail (down, for example, to cost center and account level); these figures are subject to the scrutiny of external bodies, and go on record as the legal, published results of the corporation. In contrast, the future is measured in much less detail, perhaps using somewhat different indicators such as share of market, revenue and profit growth, and similar broad indices which may take several forms or be expressed in different ways depending on the demands of the moment. If no effort is made to establish a common ground between the two approaches, each is perceived as unique and distinct, and there is no opportunity of developing a dialogue.

A second area of discontinuity concerns the methods or methodologies by which performance measures are developed. The past is measured in accordance with rigid accounting principles which reflect not only the corporation's own reporting policies, but also the requirements imposed by external bodies. However, while in this case the discipline is high, the methodologies themselves are relatively simple, and are limited in the main to elementary operations such as arithmetic addition and subtraction. Planning methodologies, on the other hand, tend to be more loosely defined, and more susceptible to change as "better" ways of coming to grips with the future are explored. Offsetting this looseness is a level of complexity far greater than that of accounting methodologies, usually as the result of attempting to emulate the workings of the real world and its nuances. In extreme cases, planning methodologies tend to the "black box" variety, consisting of "hard-wired" representations of complex cause-and-effect relationships.

Thirdly, there is the question of how the basic data of planning and accounting are defined and labelled, how they are combined with each other, and how they are displayed in the form of reports. Often a simple difference in format is interpreted as evidence of incompatibility, and serves to reinforce the apparent gap between planning and accounting. Rules for combining and displaying accounting data are rigid; coupled with a well-defined chart of accounts with specific names and codes, there is little chance of ambiguity. A chart of accounts, in fact, constitutes a ready-made set of data elements which is immediately and easily adaptable to automated processing. As with its methodologies, the data elements associated with planning are loosely defined, if at all. How they are labelled, how they are combined, is largely a personal choice, and will certainly vary from unit to unit. Quite misleading conclusions may be drawn because similarly named data elements represent different entities.

Finally, there is the area of the technical (i.e. computer) environments in which accounting and planning data are processed. The needs of accounting go back to the very beginning of electronic data processing; they have changed little since that time, continuing to reflect the orderly, structured applications that characterize accounting. Processing takes place according to regular schedules; turnaround is not usually a critical factor, and batch processing is the normal mode. The computing environment for planning applications reflects the function's demands for flexibility, ease of change, and rapid response. Typically, the environment is a time sharing one, in combination with a powerful, high-level language such as APL. Communication between the two environments is frequently difficult, due to technical reasons, with the result that the opportunity to share common data is once again lost.

Bridging the Gap

Bridging the gap means establishing an information domain within which planning and accounting can communicate in common terms. An area defined in this way need not be large, nor should it unduly inhibit the freedom of action of either side. The consequences of not building such an interface will vary depending on the general health of the corporation. If business is good and planners and accountants see very little need to talk to each other, than a bridge is probably superfluous. If, on the other hand, the marketplace is competitive, and the outlook generally volatile, there will need to be urgent and unambiguous dialogue between the two groups to provide up-to-the-minute information for tactical as well as longer-term decision making.

Constructing the bridge means taking steps to resolve, as far as possible, the problems surrounding the areas of discontinuity that were identified earlier. Associated with each of these areas is an information gap which has to be filled in some fashion. This requires taking action with respect to:

1. Data Elements
2. Methodologies
3. Formats and Conventions
4. Technical Environments

Invariably, the shortfall lies on the side of planning: accounting is a long-established, structured profession with a strict set of codes and conventions. The problem becomes one of overlaying a compatible structure on planning without destroying its flexibility or imposing a drag upon the way it does it job. Here is an account of some of the things we have been successful in putting into place:

1. Data Elements

In common with many large corporations, we use a corporate chart of accounts as the foundation for defining our corporate reporting requirements. This chart of accounts is part of our corporate policy document and reflects the combined needs of both the accounting and planning organizations at the corporate level.

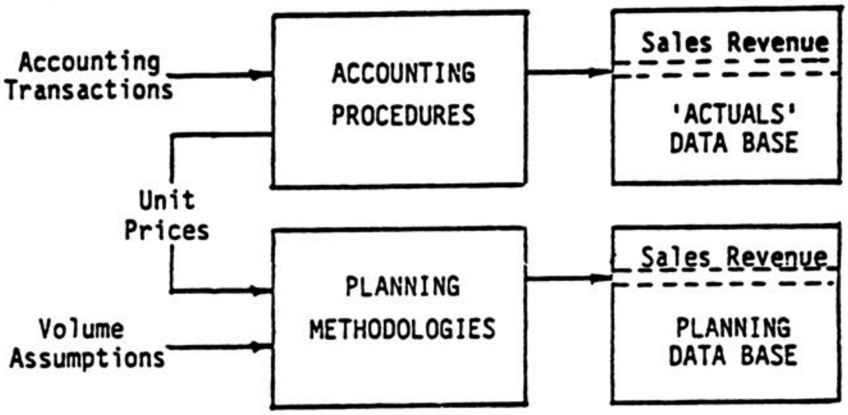
The chart is changed only on an annual basis, with the result that reclassifications, changes in reporting formats, development of new or additional data elements are held to a once-a-year exercise. Both planning and accounting are served by the same chart, except that planning elements usually (but not always) consist of aggregates of the more detailed accounting structure.

For ease of reference, and for maintenance purposes, the chart of accounts is incorporated in a live data dictionary, which plays an important role in the technical environment in which planning models and accounting systems reside. This commonality enables both sides to share the same data without the necessity of duplication in another environment, and avoids the ambiguity associated with the existence of multiple values for the same elements.

2. Methodologies

From the accounting viewpoint, methodologies are synonymous with accounting policies. Policies define the rules for the treatment of data elements from their lowest levels up to the level of corporate reporting, by which time they have been combined, aggregated, and compressed into the 500 or so items that make up the corporate chart of accounts.

Planning methodologies are also formally defined and issued as corporate policy. Their purpose is not so much to dictate what form planning models should take, but rather to ensure that model-derived values implicitly contain the ingredients that would be present in similarly-named items produced by the accounting process. For example, actual sales revenues at the corporate level consist of the aggregation of possibly thousands of individual transactions entered into ledgers around the world. These entries have been classified as "sales" in accordance with corporate policy, and might include the sale of new units, the sale of reconditioned units, and the sale of supplies.



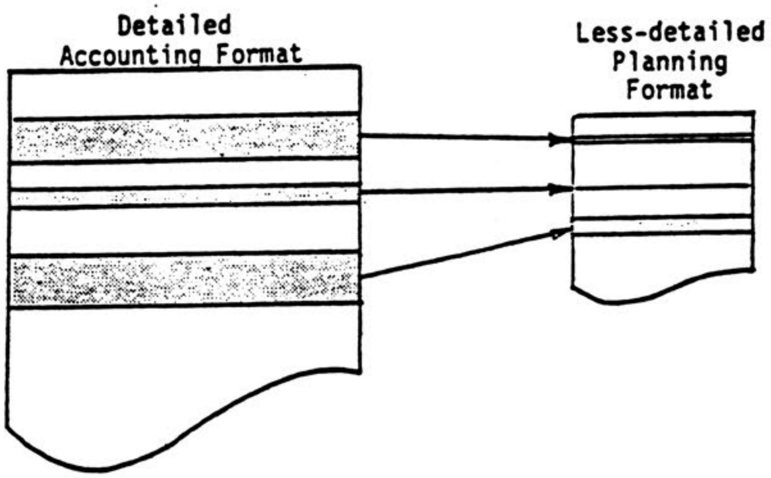
Consistency of Planning and Accounting Methodologies

Figure 2

The planning methodology paralleling this process would likely consist of the following: multiply predicted new unit sales volume by predicted unit sales price: multiply reconditioned unit sales volume by reconditioned unit sales price; perform a similar calculation for supplies, and add the three results (Figure 2). Explicit calculations of this kind serve two purposes: first, they help to ensure that in any comparison of actual to planned numbers, similar quantities are involved. Second, they enable values derived via accounting to form the basis for factor input to the planning process. There is, however, a drawback to this method: namely, it virtually dictates the use of models which, though easily understood, are by definition deterministic. This may offend those who, for one reason or another, prefer a more esoteric approach.

3. Formats and Conventions

Reports produced by the planning and accounting processes ultimately become the medium for communicating to management or to the outside world; the printed document attached to a memorandum is assumed to be "correct", and the presumption is that problems of data definition, methodologies, and so on, have all been taken care of. We have therefore been at pains to ensure that along with our chart of accounts and other policies, there is a set of corporate reports whose formats are fixed and whose line item content is defined in terms of its constituent data elements. Report logic becomes, in effect, a set of methodologies of its own.



Consistency of Accounting and Planning Report Formats

Figure 3

Recognizing that planning is carried out with less detail than accounting, we have provided alternative formats where appropriate. However, care has been taken to make certain that the less detailed report remains compatible with its longer version by making one a simple condensation of the other (Figure 3). The more detailed accounting data base can then optionally provide reports either in their expanded form, or in their condensed, planning-compatible format.

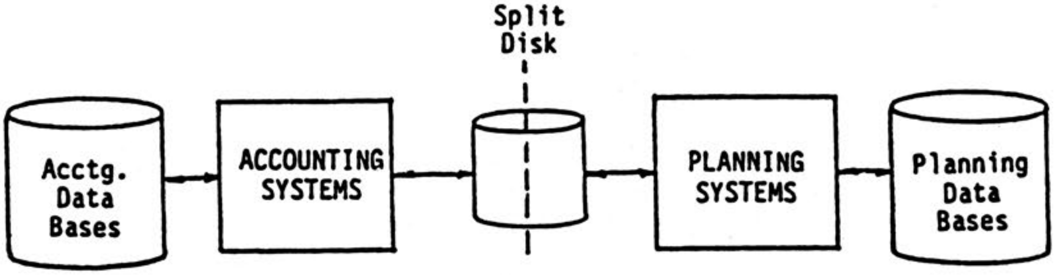
4. Technical Environment

Up to this point, the areas of change have all concerned activities outside the technical environment. However, without the appropriate capabilities in this last area, the opportunity to implement improved ways of communicating data between planning and accounting may be difficult or impossible to realize.

For a variety of reasons, any attempt to change the technical environment to meet the requirements of a relatively small community of users is hardly likely to meet with success: the inertia is simply too great. The alternative is to work in an innovative way with what is available, and to exert steady pressure to bring about change in the technical environment over time.

There is, in most cases, a barrier in communicating between a batch accounting environment and a time sharing planning environment which is as much attitudinal as it is technological. In spite of advances that have been made, and despite examples to the contrary, there remains a suspicion that any processing that takes place in a time sharing environment is not altogether serious.

By persistence, and by sheer weight of numbers, we appear to have overcome that bias, and while not having achieved full technical integration of the environments, do have the means to communicate data rapidly in both directions using shared disk technology (Figure 4). To date, this arrangement has appeared to meet all our needs in this area.



Split Disk Technology as a Means of Communication

Figure 4

Epilogue

The thesis of this paper is that competition for information between functional groups whose needs have a great deal in common is counter-productive and wasteful. Worse, the conflict may spawn the growth of other sources for the same information, which has the effect of misleading the recipients of the duplicate values.

The solution lies in establishing a minimal domain of commonality between the two groups, which typically means imposing standards and constraints upon planning staffs, who have up to this time been free to act much as they pleased. The political implications of such an action have not even been touched upon in this paper, but they may well be the greatest obstacle in bringing about any kind of improvement.

What is clear from our own experience is that this process takes time. The fact that reporting requirements — whether formalized or not — are virtually frozen over the fiscal year, means that there is but one opportunity annually to implement change, and even that must not be too radical. Success of this incremental approach is enhanced by having a long-term strategy, so that real improvements can in fact be made year after in accordance with an overall timetable. We have found that this works, and the benefits are substantial.

USING APL FOR THE QUEBEC 1980 REFERENDUM

Arthur T. Shapiro
Societe des Alcools du Quebec
Montreal, Quebec

TVA is the only independent private French T.V. network in Canada. The organization has nine member stations, with CFTM-TV in Montreal being the head of the network. This station has 57 million dollars in annual revenue and produces 76 hours of custom programming per week. Competition in Quebec is fierce because of the presence of two language groups. The networks are very conscious of their images.

During the fall of 1976, an election was announced, as usual with little warning, and the networks had less than 30 days to prepare their programs. Coverage of special events, such as an election, taxes the ability of a television station. Typical resource requirements for an event such as this are in the order of 450 persons including support staff. Props have to be designed and constructed, specialized resources have to be identified, located and briefed as to their contribution, and considerable special equipment is required. In the case of an event such as an election, there are additional and special considerations. First of all, the show must be live. Since live information has to be collected and brought to the studio on an almost instantaneous and continuous basis, a considerable amount of telecommunications gear is required. At the time of the 1976 election, I was working with Currie, Coopers and Lybrand (CCL), a management consulting firm. We were given the task of planning and organizing the data processing aspects relating to the production of the election-night program.

In view of the previous disastrous experience involving the use of shared processing arrangements for the federal election in 1974, at which time the network found itself without processing capabilities after only 20 minutes of reporting, it was decided to design a new and more reliable results-processing system.

Given the available lead time or perhaps more appropriately the lack thereof, CCL felt that APL should be used as a base language because of the reputed productivity improvement gains on other special application systems, such as the 1976 Olympic activities.

CCL had no previous experience with APL and the choice of vendor was made simply on the basis of Sharp's reputation as APL experts. Results were more than satisfactory; TVA had been badly shaken by the 1974 experience. Some of the production people had felt that a reversion to manual results processing was in order. However, in light of the resounding success of the 1976 election program, confidence was restored both in computers and in computing.

During the early part of 1980, the general feeling in the province of Quebec was that there would be a referendum before summer. TVA took advantage of the lead time and approached I.P. Sharp Associates once again to contract for results-processing services. Since referendums are hardly commonplace in Canada (the previous one in Newfoundland in 1949), it was felt that the computer programs designed for the event might possibly be out of date.

It is important to recognize from a television production programming point of view, there is an essential difference between an election and a referendum. In an election, there are several parties, several candidates, incumbents and other dignitaries, plus the problem of order of display of information within a riding or a region.

In a referendum, the riding by riding statistics serve more or less only for local information purposes. A vote basically counts for one regardless of where it is cast. The information at the riding level is normally restricted to a yes/no tally. The differences in the case of a referendum are that a consistent emphasis must be placed on consolidated (that is, total province) standing, and that individual riding information must be supplemented or built up in one way or another so as to retain viewer interest.

TVA elected to prepare socio-demographic profiles on a riding by riding basis in the interest of supplementing the live statistical information. A graduate student from the University of Montreal was retained, and proceeded to compile census and demographic information on age, income, language, ethnic origin, etc. Statistics were also gathered on the results of previous elections. The next problem involved deciding how the information should be displayed. In order to provide a high degree of viewer interest, along with innovative approaches for the referendum night, it was decided that emphasis would be placed on colour graphic techniques.

The advantage here would be twofold: first, simplicity of the display, and second, visual appeal. TVA and I.P. Sharp Associates satisfied these conditions through the use of a series of 15 APPLE II micro-computers. For programming purposes, a five second response time restraint was imposed for the construction of a graphic display. At 300 and 1200 baud, the only practical transmission speeds given the number of communication links required, it would have been absolutely impossible to construct a graph in the allotted time period—hence the need for remote intelligence.

Another reason for the choice of APPLE II equipment was its low cost and availability. We required, as was earlier stated, 15 machines for one evening, exclusive of program development and test requirements. A series of basic programs was written to allow a user to select a graphic image from a MENU, cause the APPLE II to request the appropriate data from the Toronto-based computers, and to form the image on the screen.

A total of 10 regional micro-computers were used and operated on a completely independent basis. Several days were required to develop an interface between the APPLE II and the SHARP APL system. A line out video connection between the APPLE II micro-computers and 2 colour video monitors was used to route APPLE II screen images through a manual switch. The video monitor images were captured by camera and transferred to a central control center for signal mixing. The final image consisted of a 3-part composite: the graph, reporter-analyst, and the consolidated provincial standings.

The consolidated statistics were present throughout the evening (even during commercials) and were refreshed every 5 seconds. The data for this was sent from the SHARP APL system to an output VUCOM II, and from there to a video composition device called a VIDEO IV, before transfer to the central control centre. Several accessory output devices were also used for transaction monitoring or data logging operations. Another innovation was live dynamic graph formation. This again would have been impossible using time-sharing alone.

From a response point of view, as well as from the point of view of the method of image formation (i.e. horizontal sweep) with the APPLE II computers, the histograms, which were traced on the screen, were seen to rise up directly from the base line of the graph. A total of five colour screens were used for visual effect. On the input side, a conventional and very basic data collection system was used. Thirty telephone operators recorded incoming results on slips of paper. These were handed to 10 VUCOM II key operators. The VUCOM II machines were on-line to the SHARP APL system. The data was edited and written out to file.

What now are the implications, and what lessons have been learned as a result of this experience? The 1980 Quebec referendum was a success. There are arguments as to the political implications, but from the data processing point of view, the project was a total success. Another provincial election is expected either this fall or more probably in the spring. We will definitely make extensive use of colour graphic images, controlled by micro-computers, with perhaps several innovative techniques which I will not discuss for competitive reasons.

Perhaps the most significant feature of the total approach and system was that it allowed journalists, who had absolutely no experience whatsoever in data processing, to directly control sequencing of activities within their respective regions. These people were initially apprehensive and in many cases very reluctant to take on the responsibility of operating the micro-computers, especially on the air.

After one day's worth of training, however, we detected an atmosphere of total enthusiasm. Computer use on the night of the referendum truly reflected this. While one analyst was on the air, the others, almost without exception, closely followed the progress in their ridings and conducted great numbers of simulations as a primary means of preparing themselves for their particular appearances. On a broader scale, I believe that graphics will assume more and more importance in the display of computer-generated information.

According to a recent Fortune magazine article, the manipulation of words and images as opposed to numbers will gradually assume first place. Micro-computers will play a significant, even dominant, role and applications of the election-referendum type will be commonplace.

It is my understanding that a number of hybrid systems involving use of micro-computers and the SHARP APL system have been or are being implemented by users of the service for production applications ranging from personnel management information systems, scheduling, text manipulation and data capture. This will no doubt become prominent on a more widespread basis.

APL-TIMESHARING DEVELOPMENT IN MEXICO

Daniel Olivares Guillen
Teleinformatica de Mexico, S.A.

The growth of Mexico's economy and the expansion of industry have fostered a considerable increase in the use and purchase of computers. The profile of this market and level of development reached can be observed from the following elements.

The Number of Computers Installed

In 1976 there were around 3,400 installed computers in Mexico with an annual growth not exceeding 15%. In comparison with this date, in the U.S.A. there were 150,000 computers with an annual growth rate of 25%. In relation to the gross national product, Mexico has 35.4 computers for each billion dollars, whereas the U.S.A. has 79.

The development and use of computers in Mexican industry has been influenced constantly by governmental action tending to limit imports, and to the fact that the Government itself uses at least 50% of all the computers installed in Mexico.

* Data from Nacional Financiera (NAFINSA) and the United Nations Organization for Industrial Development (ONU ID).

Users of Computers in Mexico

Three large groups of users can be identified:

1. The Government, including Semistate Companies that use 50% of the installed computers.

2. The large industrial groups, private finance companies, and

3. The private sector in general, made up of more than 300,000 small and medium size companies, of which 11,000 (3.7%) are able to use computers.

On another point, in 1976 there were more than 100 small service bureau establishments, whose joint sales did not exceed 20 million U.S. Dollars. These service-bureaus only provide batch processing services, since the operations of teleprocessing involve the use of telephone lines dedicated to data transmission, for which only two companies have the necessary authorization from the Mexican Government:

- Tiempo Compartido, S.A., a Mexican company connected to the G.E. network which does not have central equipment in Mexico, and

- Teleinformatica de Mexico, S.A., an agent of the Mexican Goverment.

The Use Given To Computers

Regarding the application of computers, it is estimated that between 50 to 70% is for administrative and commercial data processing, from 15 to 25% is for data communication, and the third place is for resolution of scientific, engineering, and educational problems.

Legal Scope

Closely related to the above, the search for new solutions in the area of Informatics is found to the conditioned by legal scope.

Through the intermediary of the Secretariat of Communications and Transport, the Mexican Government establishes that data communication by means of the telephone exchange network is not possible, and states its interest in controlling the use made for that purpose, of the country's communications infrastructure (mainly that of the telephone system). In addition, it establishes rules so that private investors can be susceptible to a licence for rendering of "The Public Service of Teleinformatics", in terms of nationality, investment and guarantees. It also establishes economic burdens for this service and regulates the tariffs.

DP Organization

Inside the organizations, the problems of using the available DP resources efficiently is characterized by the following:

1. Centralization of information processing in an area of EDP, creating bottle necks, serious communication problems between EDP experts and the users, and a lack of definition of responsibilities in the handling of the information.

2. The search for solutions with only technical criteria, impelled by the desire to have the newest of technology, which has caused a disordered growth of EDP, the loss of efficacy in favor of a badly understood technical efficiency, giving greater importance to the problem of equipment selection than to that of software, and attempting to solve all the problems by themselves, including those already resolved externally, have on occasions finally led the organization of EDP to forget the user and his real necessities.

Education

The above situation is in part caused by the lack of institutions oriented towards forming professionals in the area of informatics.

Presently, there is a small number of universities which include informatics programs in their study plans, and only one of them gives APL courses. Added to this effort is the Teleinformatica Group of Mexico which, being conscious of the need to form professionals as well as to improve communication and understanding between executives and informatics personnel, has also confronted the need to create an institute.

Teleinformatica de Mexico, S.A.

TIMSA was created on March 20, 1973, and began operations at the end of 1976. A company with 100% Mexican private capital, has the necessary authorization from the Federal Government to act as an agent of the SCT in the rendering of the "Public Service of Teleinformatics". The company's mission is to satisfy the information needs of the country's organizations through its different informatics services in an environment of scale economy, promoting the decentralization of computation, supported by the latest state of the art of teleprocessing technology, central hardware, peripheral equipment, terminals, and application software. It is also important to have a group of highly qualified experts who can "hold the bridge" between users and computer, and make the user, informatics and solutions more effectively interrelated.

TIMSA's effort in its first years has been concentrated on creating an image of alternative solution to the informatics problems of the Mexican market through the continuous incorporation of software oriented to the user, adapting the hardware to the growing needs of the market, and training of personnel.

As an example, since 1976 to the present, the central equipment has been reshaped on seven occasions, an adequate environment has been created for user education, the problem of terminals has been attacked and partially resolved, and in short, the service has been integrated vertically toward ever more developed levels.

Our services, then, are made up of various elements:

Informatics Energy which, furnished through the teleprocessing network and the terminal equipment, has its source in the central computation installation.

Application Systems and Programs oriented to three levels:

1. Information handling which includes administrative commercial systems such as payrolls, accounting, accounts payable, and others.

2. The use of Data Base Technology (ADABAS) for coordinating the information handling and information for decision making.

3. The use and applications of APL and its Public Library to support the decisions making.

Equipment and Communication. Elements that are indispensable for proper functioning of the service and that, because of the limitations of the Mexican market, we want to attack.

Consulting and Programming. For the purpose of orienting users in the use of tools, and facilitating the solution to their information problems.

Educational. Executive seminars of computer applications in companies market expansion.

Information. Through public data bases.

The History of APL

The utilization of computer "energy" in management processes is certainly something recent in Mexico. The fact that the computer facilitates decision making processes in a flexible, agile environment, is perhaps one of the greatest advantages of APL. In 1976, APL was not well known, although there were several APL installations. TIMSA, however, was and still is, the only time sharing company in Mexico that offers their users APL.

TIMSA's APL development effort has also been set in the dynamics of the company's development.

In 1976, when it was decided to start the service based on APL.SV, the challenge turned out to be more complicated considering that the problem was not only centered on making top management more aware of the use of computer tools for supporting decision making, but was also rooted in introducing the benefits of APL.SV. This, together with the concept of time sharing, made even the company's management feel it was a task for prophets.

The service was backed by a group of DP retrieval salesmen, a small support group, computer energy, the interpreter and a four-day introduction course in the use and programming of APL.

The outcome in 1977 was around 500 connect hours in the year, one or two companies in the banking fields, manufacturing and transnational companies, and one company of the group involved in Planning Consulting.

From the users' point of view, it was a small group, limited to carrying out their applications in a workspace of 120K, through 2741 terminals of 134.5 BPS, which were connected to a 370/145 with a limited maintenance support of the software, and a group of batch environment oriented operators.

In 1978, the structure of the service was re-oriented so as not only to offer the language, but also to give it an added value in Planning Consulting, which could permit us to offer a service, based both on computer capacity technology, and on an alternative solution for Top Management.

The results were satisfactory, we grew 400%. Our marketing strategy was appropriate; the panorama of our users could be generically distinguished in two groups: the first were users who had been given a solution, and a second group who learned APL as a language oriented for non-computer people that have used it and needed a solution for big and complicated projects; which were not able to develope due to the file system that implied the use of JCL language.

Considering that our market needed solutions, our application software position and marketing future was not too pleasant.

Finally, the needs and dynamics of Top Management in respect to the presentation of information were not realized in a flexible, efficient manner.

At the end of 1978, we had resolved the problems of price, speed and time delivery of terminals, and had confirmed the market profile and the capacity of APL notation; nevertheless, we questioned the growth we could have with the product, as such.

Comparative Chart Between APL.SV and SHARP APL

	USER	MARKETING	INSTALLATION	PROGRAMMER
APL.SV	- limited language - limited documentation	- no support - hardware oriented	- **poor** efficiency - IBM terminals support - **poor** documentation - **poor** support - not reliable - poor billing information - BSC and ASC support	- JCL oriented - difficulty in handling peripherals - shared variables
SHARP APL	- extended language - application software - on-line documentation	- support - user oriented - human network - continuous development - people support	- terminal support - sufficient documentation - good support - billing information - increasing efficiency - network technology - only ASC support	- N-B-T-tasks - packages - file system - report formatting - out - extended shared variables - terminal control - more primitives

At the beginning of 1979, we decided to evaluate different versions of APL, finding at this time few alternatives, these being fundamentally characterized in the following:

- Specialized APL Time Sharing Vendors, just beginning their experience as software distributors, who were concerned about the development and future of APL.

- Friendly file systems and report formatting facilities.

- Basic software was oriented to the benefit of the final user, not to the efficiency of the hardware.

- Highly reliable operation of software.

- The application software was different in each company.

- The marketing and design of application software oriented for short time solutions.

- A closed system, with no interfaces with other software or language.

TIMSA decided to take on and represent I.P. Sharp Associates with which after two months of system programmer and operators training in August of 1979, we achieved our SHARP APL system, with the major part of its application software supported by an ITEL AS-4 machine with 1 real dedicated megabyte.

The most interesting conclusion of implementation was that we had a system that was 71% more efficient.

The balance at the end of 1979 resulted in a 450% increase with respect to the previous year. TIMSA had educated 150 people in the use of the language, had achieved translation into Spanish of an APL book as a means of mass communication, and had confirmed the possibility of using APL in large projects. Together with IPSA, a Cross-Tabulation system was developed which has earned its own respect among our users.

Today, the number of our clients has grown more than 200%, and our accumulated income represents around 200% over last year.

We have begun the development of a national APL network, and we will approximately double our T-tasks and the number of terminals.

At first sight, the short term future of APL in Mexico might appear to be beyond doubt, but this is not at all cetain, as we feel there are still certain points that need to be resolved.

The decreasing costs of hardware have favoured the growth of APL installations, but more specialized and efficient application software has appeared on the market but hasn't been developed in APL.

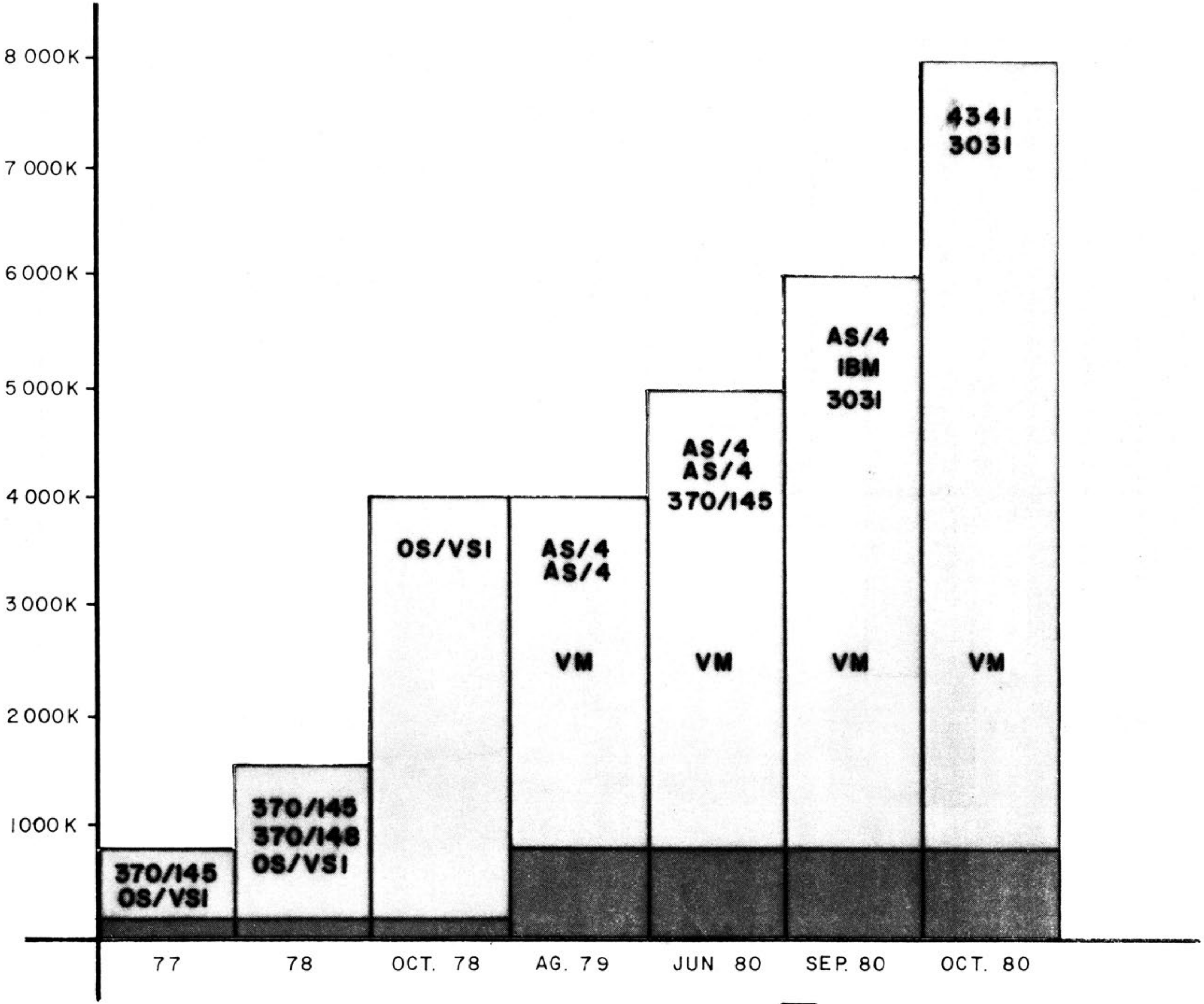
The low cost in minis and micros, as well as their blooming growth where APL has little support, will permit the growth of other languages.

The lack of interfaces with other systems restricts their field of action, at the least, as well as the lack of terminal equipment with data-entry facilities, and the fact of not having a full-screen support.

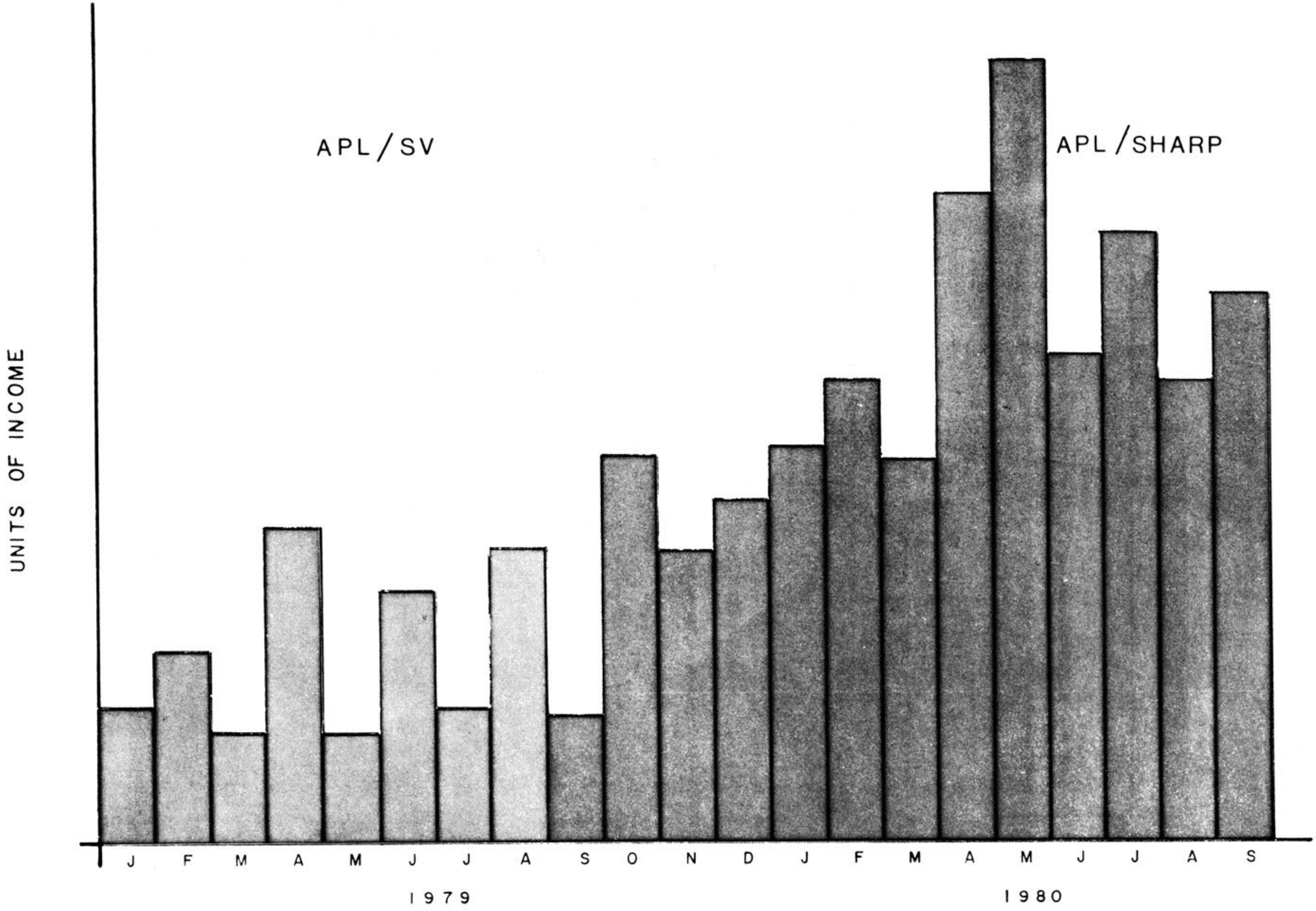
TIMSA's efforts in the future will be oriented toward greater specialization of service, with the principal objective of offering a more capable and effective tool to Top Management. We have a good product which needs educational support and better application software. We will also center our efforts on achieving an interface between APL and our ADABAS data base.

In the near future we hope to formalize a Mexican Chapter of APL users supported by STAPL, and a recognized group of APL promoters.

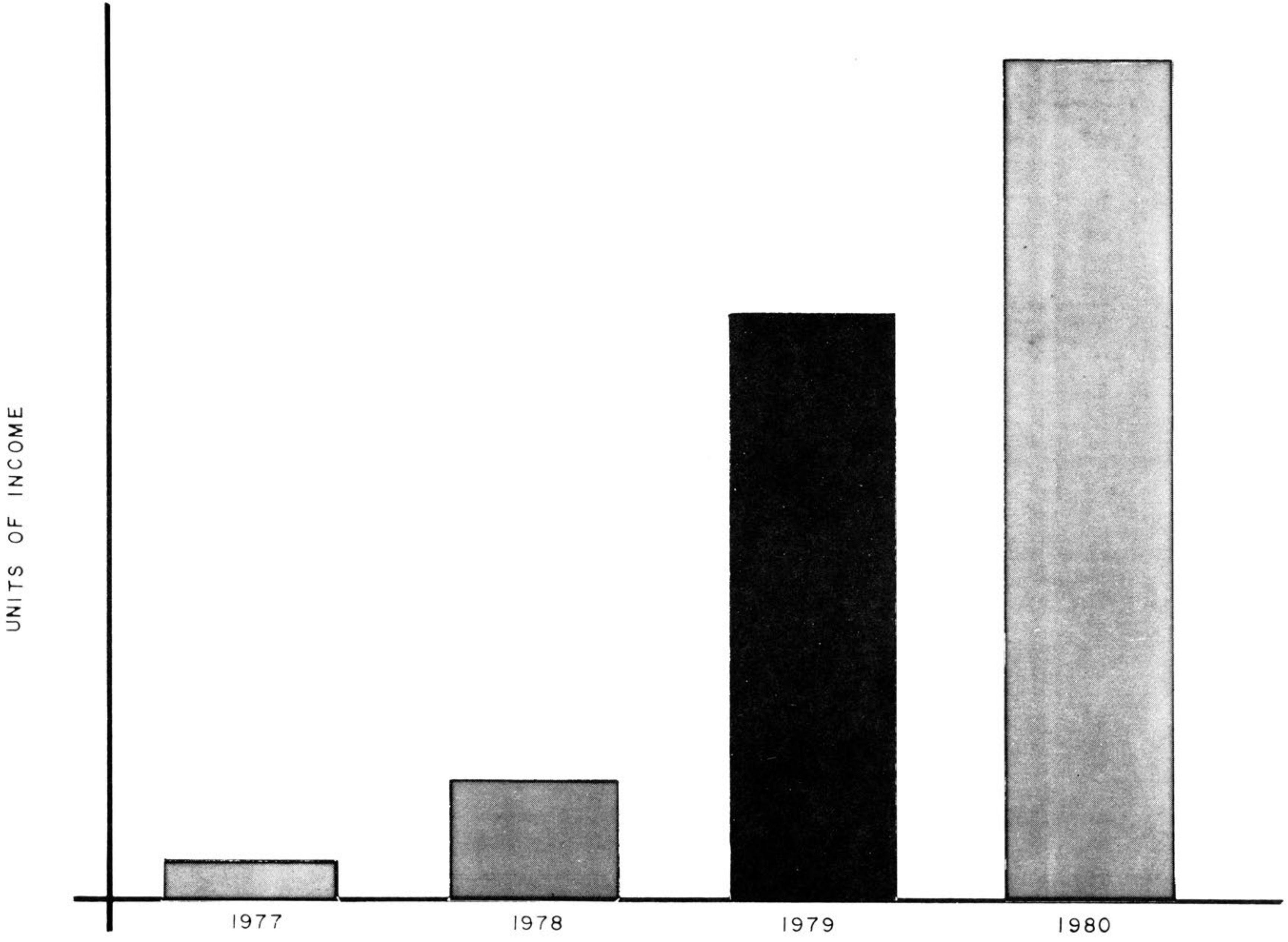
CPU'S INSTALLED



INCOME GROWTH



INCOME GROWTH


UNITS OF INCOME